LAURIBERTO SERILLO JUNIOR

UM ESTUDO COMPARATIVO ENTRE RUP E XP

Monografia apresentada a
Escola Politécnica da
Universidade de São Paulo
Para conclusão do curso de
Engenharia se software

São Paulo
2004

LAURIBERTO SERILLO JUNIOR

UM ESTUDO COMPARATIVO ENTRE RUP E XP

Monografia apresentada a
Escola Politécnica da
Universidade de São Paulo
Para conclusão do curso de
Engenharia se software

Área de Concentração:
Processos de Software
Orientador:
Prof. Dr. Kechi Hirama

São Paulo
2004

Aos meus pais, minha irmã e minha namorada que sempre me incentivam e apóiam para que eu consiga atingir meus objetivos.

AGRADECIMENTOS

Ao Prof. Dr. Kechi Hirama pela orientação, incentivo e compreensão durante a elaboração deste trabalho.

E a todos que colaboraram e incentivaram o desenvolvimento deste trabalho.

RESUMO

Este trabalho propõe um estudo comparativo entre dois processos de desenvolvimento de software o RUP e o XP visando incentivar a discussão que a escolha e a correta utilização dos processos de software são fatores determinantes para o sucesso do projeto de software. É apresentado cada um dos processos identificando suas melhores práticas e como cada um busca transformar estas práticas em benefícios para o projeto. Por fim, é feito um paralelo entre estas práticas apresentando o que os processos tem em comum e onde divergem.

ABSTRACT

This research considers a comparative study between two processes of software development: the RUP and the XP aiming at to stimulate the discussion of the choice and the correct use of software processes as determinative factors for the success of the software project. Each one of the processes is presented identifying its best practices and how it can transform these into benefits for the project. Finally, a parallel between these practices is done presenting what the processes have in common and what they differ.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 – INTRODUÇÃO

A dependência e a demanda crescentes da sociedade em relação a software, têm evidenciado uma série de problemas relacionados ao processo de desenvolvimento de software:

- alto custo;
- alta complexidade;
- dificuldade de manutenção;
- atrasos de cronograma; e
- disparidade entre as necessidades dos usuários e o produto desenvolvido.

Estes problemas existem na área de Software desde sua criação. Para administrá-los, a comunidade de Engenharia de Software, ao longo dos últimos 30 anos, estuda e implementa práticas de desenvolvimento de software bem organizadas e documentadas.

Parte do trabalho dos pesquisadores envolvidos com a Engenharia de Software tem sido descrever e abstrair modelos de processos de software. Estes modelos permitem que se compreenda o processo de desenvolvimento dentro de um paradigma conhecido. A existência de um modelo é apontada como um dos primeiros passos em direção ao gerenciamento e à melhoria do processo de software.

A maior parte dos modelos existentes é baseada em premissas tradicionais da área, uma divisão discreta das atividades do processo de software, focado em gerenciamento, medidas e registros destas atividades. Com isto surgiram vários modelos sendo que um dos mais difundidos na atualidade é o RUP (Processo Unificado Rational).

O RUP é um processo orientado ao produto (software) e não um processo de gerência de projeto. Ele contém uma coleção de modelos, diretrizes, fluxos de trabalho que ajudam os gerentes de projeto a executar diferentes tipos de projetos de

desenvolvimento de software. Além do RUP existem outros modelos, no entanto, não existe um modelo uniforme que possa descrever com precisão o que de fato acontece durante todas as fases da produção de um software; os processos implementados são muito variados, e as necessidades de cada organização diferem substancialmente.

Além disso, na última década, um segmento crescente da comunidade de Engenharia de Software vem defendendo a existência de problemas fundamentais da aplicação sistemática e institucionalizada de processos de software convencionais. Estes proponentes defendem processos simplificados, focados nas pessoas que compõem o processo, e principalmente no programador. Como diz Bach [13]: "Nas conferências e nos periódicos, a extraordinária atenção dada ao processo de desenvolvimento de software é mal dirigida. Muito se escreve sobre processos e métodos para desenvolver software; muito pouco sobre o cuidado e alimentação das mentes que de fato escrevem o software".

Eles defendem métodos conhecidos como "ágeis" ou "leves" sendo que um dos mais importantes destes métodos é o XP (*Extreme Programming*). O XP propõe uma maneira de desenvolver software onde existe uma grande preocupação em equilibrar as variáveis custo, tempo e qualidade do produto, utilizada em projetos curtos onde a variável tempo é crítica para o cliente. Para atingir estes objetivos o XP considera alguns princípios básicos: realimentação rápida, simplicidade, mudanças incrementais, adaptar-se a mudanças e trabalho de qualidade.

Alguns estudos e pesquisas que foram realizados nos anos 90 demonstraram que o gerenciamento de projeto é a causa mais evidente das falhas na execução e entrega de produtos e serviços de software. O SEI-Software Engineering Institute da Universidade de Carnegie Mellon dos EUA constatou já em 1993 que o principal problema que aflige as organizações de software é gerencial.

Um estudo conduzido pelo DoD-Department of Defense também dos EUA indicou que 75% de todos os sistemas de software falham, e que a causa principal é o fraco

gerenciamento por parte do desenvolvedor e adquirente, deixando claro que o problema não é de desempenho técnico.

O estudo realizado pelo Standish Group dos EUA, chamado de relatório do "Chaos"[1], focou a indústria de software comercial. Esse estudo identificou que: as empresas dos Estados Unidos gastaram $81 bilhões em projetos de software que foram cancelados em 1995; 31% dos projetos estudados foram cancelados antes de serem concluídos; 53% dos projetos de software que foram concluídos excederam mais do que 50% a sua estimativa de custo; somente 9% dos projetos, em grandes empresas de desenvolvimento de software, foram entregues no tempo e orçamento.

Portanto entender as principais vertentes de processos e qual melhor se adapta ao projeto e a corporação são fatores importantes para o sucesso dos projetos.

1.1 - Objetivo

Este trabalho tem por objetivo apresentar dois dos mais difundidos processos de desenvolvimento de software e fazer um estudo comparativo entre eles. O objetivo é oferecer subsídios para a escolha de um processo que seja adequado ao tipo de projeto e grau de maturidade da corporação e ainda apresentar as boas práticas de cada um deles.

1.2 - Motivações

Cada vez mais os projetos de software são exigidos nas questões de prazo, custo, qualidade e manutenção. Isto nos remete a uma série de incertezas tais como:

- O projeto produzirá um produto satisfatório?
- O projeto produzirá um produto de qualidade?
- O projeto terminará no prazo?
- Será necessário trabalho além do previsto?
- Os compromissos com o cliente serão cumpridos?

Estas incertezas nos motivam a utilizar processos para gerenciar nossas atividades e com isto minimizar estes riscos.

Porém muitos estudos mostraram que a maioria dos fatores que afetam o desenvolvimento de software são falhas específicas no domínio do gerenciamento de projeto, ou estão diretamente associados às práticas de gerenciamento mal empregadas. Isto mostra que além de utilizar processos de gerenciamento é importante escolher qual melhor se adapte à cada projeto. Portanto, a motivação deste trabalho é apresentar duas diferentes vertentes de processos de software, ou seja, métodos ágeis ("leves") e métodos tradicionais ("pesados") através de seus dois mais difundidos exemplos, para com isto ajudar na escolha e aplicação do processo adequado para cada situação.

1.3 – Estrutura do trabalho

Este trabalho está dividido nos seguintes capítulos:

- O capítulo 1 descreve o objetivo e as motivações do trabalho.
- O capítulo 2 apresenta as definições e abreviaturas utilizadas no trabalho.
- O capítulo 3 descreve o Processo Unificado Rational apresentando sua forma de desenvolvimento, fases e fluxo de trabalho.
- O capítulo 4 descreve *Extreme Programming* apresentando suas dimensões, ciclo de vida e as atividades executadas durante o ciclo de vida.
- O capítulo 5 faz uma comparação entre os dois processos apresentados anteriormente tomando como ponto base os princípios de *Extreme Programming.*
- O capítulo 6 descreve as conclusões do trabalho, as contribuições e uma proposta de trabalhos futuros.
- No capítulo 7 encontram-se as referências bibliográficas utilizadas no trabalho.

2 – DEFINIÇÕES E ABREVIATURAS

Ator	Um conjunto coerente de papéis que os usuários de casos de uso desempenham ao interagir com os casos de uso.
Casos de Uso	A descrição de um conjunto de seqüências de ações, incluindo variantes, que um sistema realiza, fornecendo o resultado observável do valor de um ator.
Cartões CRC	(Classe Responsabilidade Colaboração) Cartões CRC são utilizados para gravar as responsabilidades e colaboradores das classes.
Cenário	Uma seqüência específica de ações que ilustram o comportamento.
Classes	A descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica.
Estórias	Algo que o cliente deseje que o sistema faça. As estórias podem ser testadas.
Modelo de objetos	Um modelo que mostra um conjunto de objetos e seus relacionamentos em um ponto no tempo; os modelos de objetos abrangem a visão estática de projeto ou a visão estática de processo de um sistema.
Propriedade coletiva	É a equipe como um todo sendo responsável por cada arquivo de código. Qualquer membro da equipe pode alterar o código.
Stakeholders	Alguma pessoa ou representante de uma organização que tem interesse no resultado de um projeto. Um *stakeholder* pode ser um usuário final, um comprador, um desenvolvedor ou um gerente de projeto.
UML	(Unified Modeling Language) Linguagem de Modelagem Unificada, uma linguagem para a visualização, especificação, construção e documentação de artefatos de um sistema complexo de software.

3 - RUP – PROCESSO UNIFICADO RATIONAL

3.1 – Introdução ao RUP

O Processo Unificado Rational (RUP) é um processo de engenharia de software que provê uma forma disciplinada de definir as tarefas e responsabilidades dentro de uma organização de desenvolvimento de software. Sua meta é assegurar a produção de alta qualidade de software para que este satisfaça as necessidades de seus usuários, dentro de prazo e custo previsíveis. [4].

O Processo Unificado Rational (RUP) procura aumentar a produtividade da equipe, proporcionando para todos membros fácil acesso à uma base de conhecimento com diretrizes, modelos e ferramentas para todas as atividades críticas do desenvolvimento. Para que todos tenham acesso à mesma base de conhecimento, não importando se trabalha com requisitos, projeto, teste, gerenciamento do projeto, ou gerencia de configuração, o RUP tenta assegurar que todos os membros da equipe compartilham uma linguagem comum e uma visão de como desenvolver software. As atividades do RUP criam e mantêm modelos. Em lugar de produzir uma grande quantia de documentos, o RUP enfatiza o desenvolvimento e manutenção de modelos.

O RUP é um guia de como usar efetivamente a Linguagem Unificada de Modelagem (UML). A UML é uma linguagem padrão que nos permite comunicar claramente requisitos, arquiteturas e projeto. O Processo Unificado Rational é apoiado por ferramentas que automatizam grandes partes do processo. Elas são usadas para criar e manter os vários artefatos do processo: modelagem visual, programação, teste, etc. Estas ferramentas são importantes para apoiar o gerenciamento de mudanças e o gerenciamento de configuração de cada iteração.

O Processo Unificado Rational é um processo configurável. Ele se ajusta a pequenas equipes de desenvolvimento como também grandes organizações de desenvolvimento. O Processo Unificado Rational é baseado em uma arquitetura de

processo simples e clara através de um conjunto de processos. E estas podem ser modificadas para acomodar as diferentes situações das organizações.

O Processo Unificado Rational captura muitas das melhores práticas do desenvolvimento de software, em uma forma que é satisfatória para uma gama extensiva de projetos e organizações.

3.2 - Desenvolvimento efetivo com RUP

O Processo Unificado Rational descreve como utilizar práticas efetivamente comprovadas para as equipes de desenvolvimento de software. Estas são chamadas "melhores práticas" não por poder quantificar o seu valor, mas porque elas são utilizadas com sucesso nas organizações. O Processo Unificado Rational provê diretrizes, modelos e ferramentas para que a equipe tire proveito das práticas:[2]

1. Desenvolvimento iterativo
2. Gerencia de requisitos
3. Uso de arquiteturas baseadas em componentes
4. Modelagem visual do software
5. Verificação da qualidade do software
6. Gerenciamento de mudanças do software

A seguir, as práticas são descritas com mais detalhes.

3.2.1 - Desenvolvimento iterativo

Dados os sistemas de software sofisticados de hoje, não é possível executar de uma forma seqüencial, primeiro definir o problema inteiro, projetar a solução inteira, construir o software e então testar o produto todo. É necessária uma forma iterativa que permita a compreensão crescente do problema por sucessivos refinamentos desenvolvendo uma solução efetiva através de múltiplas iterações.

O RUP sugere uma forma iterativa para o desenvolvimento onde estas iterações tratam os riscos mais altos em todas as fases no ciclo de vida, buscando assim reduzir o risco do projeto. Estas iterações minimizam os riscos através de liberações freqüentes, executáveis que propiciam envolvimento do usuário e a contínua

avaliação do produto. Cada iteração termina com a liberação de um executável. A equipe de desenvolvimento está focalizada em produzir resultados, e freqüentemente estes resultados são conferidos a fim de assegurar que o projeto fique dentro dos prazos acordados. Com esta forma iterativa também se torna mais fácil efetuar mudanças de requisitos, características ou cronograma.

3.2.2 - Gerencia de requisitos

O Processo Unificado Rational descreve como extrair, organizar, documentar requisitos e funcionalidades; rastrear mudanças de documentos e decisões; e facilmente capturar e comunicar requisitos de negócio. As noções de caso de uso e cenários descritos no processo provaram ser um excelente modo para capturar requisitos funcionais e assegurar que estes conduzem o projeto, implementação e teste de software, buscando assim que o sistema final cumpra as necessidades solicitadas pelo usuário[2].

3.2.3 - Uso de arquiteturas baseadas em componentes

O processo se orienta em desenvolver uma arquitetura base, antes de executar qualquer desenvolvimento completo. Descreve como projetar uma arquitetura flexível, que aceita mudanças, é intuitivamente compreensível, e promove o reuso do software. O RUP apóia o desenvolvimento de software baseado em componentes. Estes componentes são módulos, subsistema que cumprem uma função clara.

O Processo Unificado Rational provê um modo sistemático para definir uma arquitetura que usa componentes novos e existentes. Oferecendo modelos para descrever a arquitetura baseado no conceito de múltiplas visões da arquitetura, e provendo diretrizes para a escolha da arquitetura.

Os componentes são agrupados dentro uma arquitetura bem definida, ou em uma infra-estrutura de componente como a Internet, CORBA, e COM, para qual a industria de componentes reutilizáveis esteja emergindo.[2]

3.2.4 - Modelagem visual do software

O processo mostra como a modelagem visual captura a estrutura e o comportamento de arquiteturas de componentes. Isto permite abstrair os detalhes e escrever código usando blocos de construção gráfica.

Estas abstrações visuais ajudam a comunicar aspectos diferentes do software; vendo como os elementos do sistema se ajustam, estes blocos de construção são consistentes com seu código; mantendo assim consistência entre um projeto e sua implementação, propiciando uma comunicação continua entre todas as fases e pessoas de modo que não se torne ambígua [5].

3.2.5 - Verificação da qualidade do software

Baixo desempenho da aplicação e baixa confiabilidade são fatores comuns que inibem a aceitação das aplicações de software de hoje. Conseqüentemente, a qualidade deveria ser revisada com respeito aos requisitos baseada em confiabilidade, funcionalidade e desempenho do sistema. Verificar a qualidade soluciona vários problemas do desenvolvimento de software:

- A avaliação de projeto deixa de ser subjetiva por ser feita com base nos resultados dos testes e não em documentos;
- Esta avaliação tem por objetivo mostrar as inconsistências nos requisitos e a sua implementação;
- Os testes são focados em áreas de alto risco aumentando a qualidade e diminuindo estes riscos;
- Os erros são identificados mais rapidamente, reduzindo assim o custo de sua correção; e
- O uso de ferramentas automatizadas auxiliam na avaliação das funcionalidades, o desempenho e a escalabilidade.

No RUP a avaliação de qualidade é realizada, em todas as atividades, envolvendo todos os participantes, usando objetivo, métricas e critérios, e não como uma atividade á parte executada posteriormente por uma equipe separada.

3.4 - Fases e Iterações

A organização dinâmica do Processo Unificado Rational está dividida em fases e iterações ao longo do tempo.

O ciclo de vida do software está dividido em ciclos, cada ciclo evolui para uma nova versão do produto. O RUP divide um ciclo de desenvolvimento em quatro fases sucessivas:

- Concepção
- Elaboração
- Construção
- Transição

Cada fase é concluída quando um marco bem definido no qual decisões críticas e metas fundamentais devem ter sido alcançadas.

Os maiores marcos do projeto estão ao final de cada fase, ou seja, ao final da fase de concepção, Elaboração, Construção e Transição conforme a Figura 2 [3].

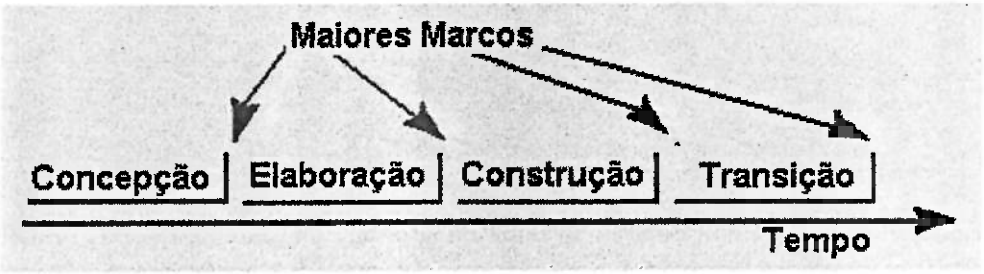

Figura 2 – Maiores marcos do RUP.

3.4.1 - Concepção

Durante a fase de Concepção, estabelece-se o caso de uso de negócio para o sistema e delimita-se o escopo do projeto. Para isto devem ser identificadas as entidades externas que o sistema interagirá (atores) e a natureza desta interação em um alto-nível. Isto envolve identificar os casos de uso.

O caso de uso de negócio inclui critérios de sucesso, a avaliação de riscos, a estimativa de recursos necessários e um plano de gerenciamento da fase, mostrando a previsão dos principais marcos de progresso.

Durante a fase de Concepção é comum a criação de um protótipo executável, servindo como teste para a Concepção. No final da fase de Concepção, os objetivos do ciclo de vida do projeto devem ser examinados e decide se deve prosseguir para a próxima fase.

Os resultado da fase de Concepção são [4]:

- Um documento de visão: uma visão geral dos requisitos e da essência do projeto, características fundamentais, e principais restrições;
- Um modelo de caso de uso de 10% a 20% completo;
- Um glossário inicial do projeto;
- Um caso uso de negócio inicial que inclui contexto empresarial, critérios de sucesso (projeção de renda, mercado, reconhecimento, e assim por diante), e previsão financeira;
- Uma avaliação de risco inicial;
- Um plano de projeto, mostrando fases e iterações;
- Um modelo de negócio, se necessário; e
- Um ou vários protótipos.

Ao término da fase de Concepção está o primeiro marco principal do projeto. Os critérios de avaliação para a fase de concepção são:

- O consentimento dos *Stakeholders* na definição do escopo e estimativas de custo e prazo;
- Requisitos entendidos e evidenciados com fidelidade nos casos de uso primários;
- Credibilidade das estimativas de custo, prazo, prioridades, riscos, e processo de desenvolvimento;
- Profundidade e amplitude de qualquer protótipo da arquitetura que tenha sido desenvolvido; e

- Revisão das despesas atuais contra despesas planejadas.

O projeto pode ser cancelado ou consideravelmente re-planejado se não passar por este marco.

3.4.2 - Elaboração

As metas da fase de Elaboração são a análise do escopo do problema, o estabelecimento de uma arquitetura bem definida, o desenvolvimento do plano do projeto e a eliminação dos elementos de mais alto risco do projeto. As decisões de arquitetura devem ser feitas com uma compreensão de todo o sistema. Isso significa o entendimento do sistema inteiro: seu escopo, funcionalidades principais e requisitos não funcionais como requisitos de desempenho.

A fase de Elaboração é a mais crítica das quatro fases. Ao término desta fase, grande parte da engenharia é considerada completa e deve ser decidido se é iniciada ou não a fase de Construção e Transição. Para a maioria dos projetos, isto corresponde também para a transição de uma operação ágil, de pouco risco para um alto-custo, operação de alto risco com significativa inércia. Enquanto o processo sempre tiver que acomodar mudanças, as atividades da fase de Elaboração asseguram que a arquitetura, requisitos e planos são bastante estáveis, e os riscos são suficientemente minimizados. Assim, de maneira previsível é possível determinar o custo e programar a conclusão do desenvolvimento. Conceitualmente, este nível de fidelidade corresponderia ao nível necessário para uma organização iniciar a fase de Construção a um preço fixo.

Na fase de Elaboração, um protótipo de arquitetura executável é construído em uma ou mais iterações, dependendo do escopo, tamanho, risco, e novidade do projeto.

Este esforço deveria atingir os casos de uso críticos identificados pelo menos na fase de Concepção que tipicamente expõe os riscos técnicos principais do projeto. Enquanto um protótipo evolutivo de um componente de qualidade sempre é a meta, isto não exclui o desenvolvimento de um ou mais protótipos para minimizar riscos

específicos como mudanças de projeto e requisitos, estudo de viabilidade de componente, ou demonstrações para investidores, clientes, e usuários.

O resultado da fase de Elaboração é:

- Um modelo de caso de uso (com pelo menos 80% definido) onde todos casos de uso e os atores foram identificados, e a maioria do casos de uso foram descritos;
- Requisitos adicionais que capturam requisitos não funcionais e alguns requisitos que não estão associados com caso de uso específico levantados e documentados;
- Uma descrição da arquitetura de software;
- Um protótipo arquitetônico executável;
- Uma lista de risco revisada e um caso de negócio revisado;
- Um plano de desenvolvimento para o projeto global, inclusive o plano de projeto, que mostra as iterações e critérios de avaliação para cada uma;
- Um caso de desenvolvimento atualizado que especifica o processo a ser usado; e
- Um manual preliminar do usuário (opcional).

O término da fase de Elaboração é o segundo marco importante do projeto. Neste momento, são examinados os objetivos detalhados e o escopo do sistema, a escolha de arquitetura e a resolução dos riscos.

Os critérios de avaliação para a fase de Elaboração envolvem as questões:

- A visão do produto é estável?
- A arquitetura é estável?
- O protótipo mostra que os principais elementos de risco foram solucionados?
- O plano de gerenciamento para a fase de construção está suficientemente detalhado e preciso? É apoiado com base em estimativas?

- Todos os *stakeholders* concordam que a visão atual pode ser alcançada se o plano atual for executado para desenvolver o sistema completo, no contexto da arquitetura atual?
- As despesas e os recursos atuais estão de acordo com as despesas planejadas?

O projeto pode ser abortado ou consideravelmente re-planejado se não passar por este marco.

3.4.3 - Construção

Durante a fase de Construção, é desenvolvido de maneira iterativa e incremental, um produto completo e todas as suas características são testadas completamente. A fase de Construção é um processo industrial onde é dado ênfase em administrar recursos e controlar operações para aperfeiçoar custos, prazos e qualidade.

Muitos projetos são grandes o bastante e podem ser gerados incrementos de construção paralelos. Estas atividades paralelas podem apressar significativamente as liberações; eles também podem aumentar a complexidade da administração dos recursos e sincronização de fluxo de trabalho. Uma arquitetura robusta e um plano compreensível são altamente correlatos.

Em outras palavras, uma das qualidades críticas da arquitetura é sua facilidade de construção. Esta é a razão por que é dado ênfase ao desenvolvimento da arquitetura e o plano durante a fase de Elaboração. O resultado da fase de Construção é um produto pronto para o usuário final. O conjunto mínimo, desta fase consiste em:

- O produto de software interage corretamente nas plataformas adequadas;
- Os manuais de usuário descritos; e
- Uma descrição da liberação atual.

O término da fase de Construção é o terceiro marco principal do projeto. Neste momento, é decidido se o software, a infra-estrutura, e os usuários estão prontos para

se tornar operacionais, sem expor o projeto a altos riscos. Esta liberação é chamada freqüentemente de versão beta.

Os critérios de avaliação para a fase de construção envolvem as questões:

- O produto liberado é estável e maduro o bastante para ser instalado na comunidade usuária?
- Todos os *stakeholders* estão prontos para a transição na comunidade usuária?
- As despesas e os recursos atuais estão de acordo com as despesas planejadas?

A fase de Transição pode ter que ser adiada se o projeto não alcançar este marco.

3.4.4 - Transição

O propósito desta fase é a transição do produto de software para a comunidade usuária. Uma vez que o produto tenha sido dado ao usuário, assuntos novos normalmente surgem e requerem algum desenvolvimento adicional, com a finalidade de ajustar o sistema, corrigir alguns problemas identificados, ou terminando as características que foram adiadas.

A fase de Transição é iniciada quando uma versão beta for madura o bastante para ser utilizada no domínio do usuário. Isto requer tipicamente que algum subconjunto utilizável do sistema esteja completo, a um nível aceitável de qualidade e a documentação de usuário esteja disponível de forma que a transição proverá resultados positivos para todas as partes.

Esta fase inclui:

- Teste beta, valida o sistema novo contra as expectativas dos usuários;
- Operação paralela com um sistema legado que está sendo substituído;
- Conversão de bancos de dados operacionais;
- Treinamento de usuários;e

- Implantação do produto para o marketing, distribuição e equipes de vendas.

A fase de Transição focaliza nas atividades exigidas para colocar o software nas mãos dos usuários. Tipicamente, esta fase inclui várias iterações, com liberações de versão beta, como também liberação de correção de versões com problema. É gasto um esforço considerável desenvolvendo documentação orientada a usuário, treinando os usuários, apoiando o uso inicial do produto pelos usuários. Porém, neste momento no ciclo de vida a avaliação dos usuários deveria ser principalmente limitada a ajustar o produto, quanto à configuração, instalação, e usabilidade da versão.

Os objetivos primários da fase de Transição incluem:

- Oferecer suporte adequado aos usuários; e
- Conseguir o consentimento dos *stakeholders* que o produto desenvolvido esta completo e consistente com os critérios de validação estabelecidos.

Dependendo da complexidade do produto esta fase também pode ser simples ou complexa. O término da fase de Transição é o quarto marco mais importante do projeto, o marco de liberação do produto.

Neste momento é decidido se os objetivos foram alcançados, e se deve ser iniciado outro ciclo de desenvolvimento. Em alguns casos, este marco pode coincidir com o fim da fase de Concepção do próximo ciclo.

Os critérios de avaliação primários para a fase de Transição envolvem as envolvem as questões:

- o usuário está satisfeito?
- As despesas e os recursos atuais estão de acordo com as despesas planejadas?

3.4.5 - Iterações

Cada fase do RUP ainda pode ser dividida em iterações. Uma iteração é um ciclo completo de desenvolvimento, resultando em uma versão de um produto executável que constitui um subconjunto do produto final em desenvolvimento. Esta versão pode ser interna, isto é, não é liberada para o usuário final ou externa neste caso é liberada para o usuário final. Esta versão cresce de modo incremental de uma iteração para outra até se tornar o sistema final. Cada iteração passa pelos vários fluxos de trabalho do processo, embora com uma ênfase diferente em cada um deles, dependendo da fase.

Durante a fase de Concepção, o foco está na captação de requisitos. Durante a Elaboração o foco passa a ser a análise e o projeto. A implementação é a atividade central na Construção e a entrega é o foco da Transição.

3.4.6 - Benefícios de um processo iterativo

Comparado ao processo tradicional cascata, o processo iterativo tem as seguintes vantagens:

- Os riscos são minimizados mais cedo;
- As mudanças são mais fáceis;
- Existe um nível mais alto de reutilização;
- A equipe do projeto pode aprender ao longo do caminho; e
- Melhor qualidade global..

3.5 - Estrutura estática do processo

Todo processo deve responder a pergunta: Quem está fazendo o que, quando e como? O RUP representa isto usando 4 elementos primários (Figura 3) :

- Papéis: "quem"
- Atividades: "como"
- Artefatos: "o que"
- Fluxo de trabalho: "quando"

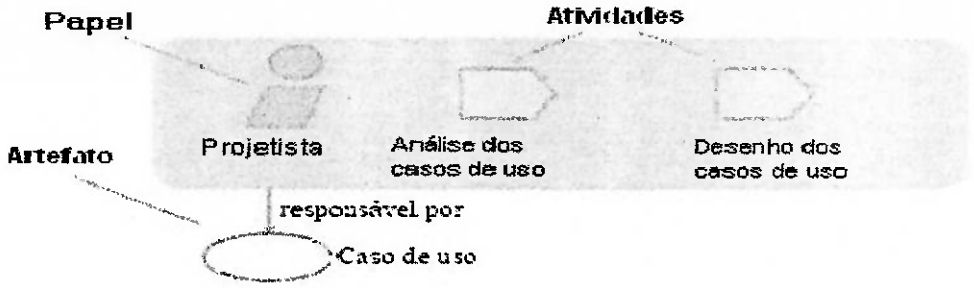

Figura 3 – Estrutura estática do RUP.

3.5.1 - Papéis

Um papel define o comportamento e as responsabilidades de um indivíduo, ou um grupo que trabalha como uma equipe. Um indivíduo pode desempenhar mais que um papel. As responsabilidades atribuídas a um papel incluem tanto a execução de um conjunto de atividades como a titularidade de certos artefatos.

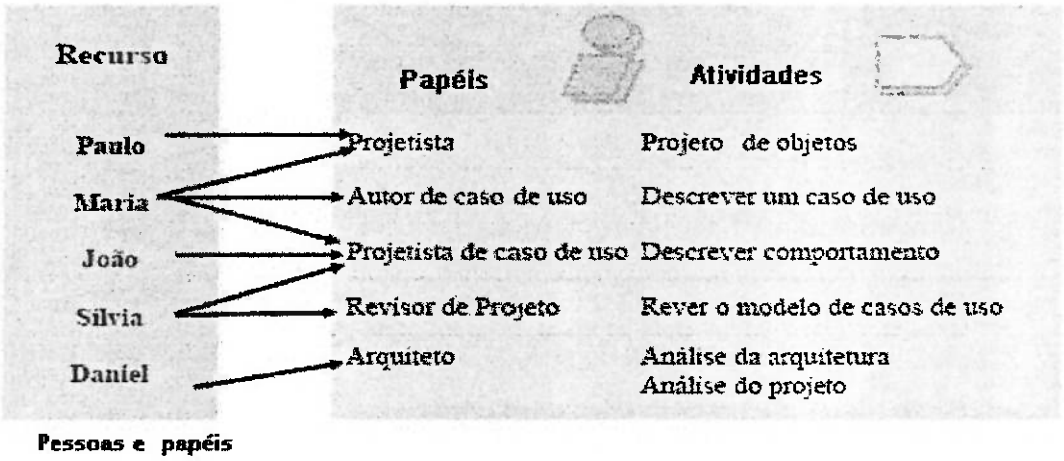

Figura 4 – Papéis no RUP.

3.5.2 - Atividades

Uma atividade é uma unidade de trabalho que pode ser solicitada a um indivíduo num dado papel. Tem um objetivo claro, geralmente expresso em termos de produção ou atualização de um artefato (modelo, classe, plano...). Todas as atividades são atribuídas a um papel específico. A granularidade pode ser de horas a dias, normalmente envolve um papel e afeta um artefato ou um conjunto reduzido deles.

Deve ser utilizável como elemento de planejamento e aferição de progresso. Exemplos de atividades:

- Planejar uma iteração (papel: gestor do projeto);
- Identificar casos uso e atores (papel analista);
- Rever o projeto (papel: revisor de projeto); e
- Executar testes de desempenho (papel: testador).

3.5.3 - Artefatos

É o elemento (de informação) produzido, modificado ou utilizado por um processo. São os elementos tangíveis de um projeto (aquilo que são produzidos ou utilizados ao trabalhar para o objetivo). São utilizados como entrada pelos indivíduos (papéis) para desempenhar uma atividade e são o resultado ou saída de tais atividades. Exemplos:

- Um modelo de casos de uso;
- Um elemento de um modelo – uma classe;
- Um documento –caso de negócio ou arquitetura do software;
- Código fonte; e
- Códigos executáveis (binários).

3.5.4 - Fluxo de trabalho

Os papéis, atividades e artefatos não constituem um processo totalmente. É preciso um modo para descrever seqüências significantes de atividades que produzem algum resultado, e mostrar interações entre os papéis.

Um fluxo de trabalho é uma seqüência de atividades que produzem um resultado de valor observável. Em UML, um fluxo de trabalho pode ser expresso como um diagrama de seqüência, um diagrama de colaboração ou um diagrama de atividade.

Nem sempre é possível ou prático representar todas dependências entre atividades. Freqüentemente duas atividades são entrelaçadas mais firmemente, especialmente quando envolvem o mesmo trabalhador ou o mesmo papel. As pessoas não são máquinas, e o fluxo de trabalho não pode ser interpretado literalmente como um programa para as pessoas seguirem mecanicamente.

3.6 – Divisão do fluxo de trabalho no RUP

Há nove fluxos de trabalho descritos no RUP eles representam a divisão de atividades entre os indivíduos (papéis) agrupados logicamente[4].

O fluxo de trabalho do processo é dividido em seis fluxos de trabalho de engenharia:

1. Modelagem de negócio
2. Requisitos
3. Análise e projeto
4. Implementação
5. Teste
6. Entrega

E três fluxos de trabalho de suporte

1. Gerenciamento do projeto
2. Gerenciamento de configuração
3. Ambiente

Embora os nomes dos seis fluxos de trabalho de engenharia possam parecer às fases de um processo cascata tradicional, as fases do processo iterativo são diferentes e que estes fluxos de trabalho são executados repetidas vezes ao longo do ciclo de vida. O fluxo de trabalho completo atual de um projeto intercala estes nove fluxos, e os repete com várias ênfases e intensidades a cada iteração.

3.6.1 - Modelagem de negócio

Um dos problemas principais com a modelagem de negócio, é que a comunidade que desenvolve o software e a comunidade que cria o negócio não se comunicam corretamente.. O RUP tenta minimizar isto provendo um idioma comum e processos para ambas as comunidades, como também mostrando como criar e manter a rastreabilidade direta entre negócio e modelos de software.

Na modelagem de negócio documentam-se processos empresariais chamados casos de uso de negócio. Isto assegura um entendimento comum entre todo os *stakeholders* do processo de negócio precisa ser suportado na organização.

Os casos de uso de negócio são analisados para entender como a empresa deveria suportar os processos de negócio. A modelagem de negócio consiste em duas partes principais: um modelo caso de uso empresarial e um modelo de objetos empresarial, ambos podem ser criados utilizando a UML. O modelo de caso de uso empresarial descreve os processos empresariais e são documentados como uma sucessão de ações que provêem valor a um ator empresarial. O modelo de objetos empresarial mostra como o modelo de caso de uso de negócio será realizado. Ele serve como uma abstração de como os atores e entidades de negócio precisam se relacionar e colaborar para executar o negócio. Muitos projetos podem escolher não fazer esta modelagem do negócio caso o produto a ser gerado não esteja fortemente embutido dentro de algum processo empresarial. [14]

2.6.2 - Requisitos

A meta do fluxo de trabalho de requisitos é descrever o que o sistema deveria fazer e permitir que desenvolvedores e clientes concordem com a descrição. Para alcançar isto, extraem-se, organizam-se, e documentam-se requisitos funcionais e restrições; rastreiam-se mudanças de documentos e decisões. Um documento de visão é criado e as necessidades dos *stakeholder* são extraídas. São identificados os atores, representando os usuários, e qualquer outro sistema com o que o sistema que esta sendo desenvolvido possa interagir. São identificados casos de uso, representando o comportamento do sistema.

Cada caso de uso é descrito em detalhes. A descrição do caso de uso mostra como o sistema interage passo por passo com os atores e o que o sistema faz. São descritos requisitos não funcionais em especificações adicionais. O mesmo caso de uso é usado durante a captura de requisitos, análise e projeto e teste.

3.6.3 - Análise e projeto

A meta do fluxo de trabalho da análise e projeto é mostrar como o sistema será implementado. O sistema a ser construído necessita que:

- Execute em um ambiente específico as tarefas e funções especificadas nos casos de uso;
- Atenda todos os requisitos; e
- Seja estruturado para ser robusto (fácil de mudar quando os requisitos funcionais mudarem).

Da análise e projeto resulta em um modelo de projeto e um modelo de análise. O modelo de análise consiste em um modelo de objetos que descreve a realização dos casos de uso e serve como uma abstração do modelo de projeto. Normalmente este modelo de objetos evoluirá diretamente nos elementos do modelo do projeto, sendo que a meta do modelo de análise é fazer uma mapeamento preliminar do comportamento exigido pelo sistema. A meta do modelo de projeto é transformar este modelo preliminar transformando em elementos que podem ser implementados. Portanto, há um refinamento de detalhes quando se passa do modelo de análise para o modelo de projeto. Portanto o modelo de análise passa por muitas alterações e com isto se torna opcional fazer um modelo de análise ou já se construir diretamente o modelo de projeto. Existem alguns pontos que devem ser considerados para saber se o modelo de análise pode ser opcional ou não:

- O projeto é complexo, tal que é necessário uma abstração para apresentar o mesmo a todos os membros da equipe;
- O trabalho gasto para se manter o sincronismo entre o modelo de análise e o modelo de projeto esta justificando a necessidade de manter os artefatos com diferentes níveis de abstração; e
- Em algumas organizações onde os sistemas te um longo tempo de vida, ou onde há muitas variantes do sistema, um modelo de análise pode ser útil.

O modelo de projeto serve como uma abstração do código fonte, isto é, o modelo de projeto descreve como o código fonte é estruturado e escrito.

O modelo de projeto consiste em classes de projeto estruturadas em pacotes de projeto e subsistema com interfaces bem definidas, representando o que se tornarão componentes na implementação. Também contém descrições de como estas classes de projeto colaboram para executar os casos de uso.

As atividades de análise e projeto são centradas na arquitetura. A produção e validação desta arquitetura é o foco principal das iterações desta fase. A arquitetura é representada por várias visões [6]. Estas visões capturam as principais decisões de projeto da estrutura.

Em essência, visões de arquitetura são abstrações ou simplificações do projeto inteiro no qual características importantes são criadas deixando detalhes à parte. A arquitetura não só é um veículo importante para desenvolver um bom modelo de projeto mas também para aumentar a qualidade de qualquer modelo construído durante o desenvolvimento do sistema.

3.6.4 - Implementação

O propósito do fluxo de trabalho de implementação é:

- Definir a organização do código, em termos de subsistemas de implementação organizados em camadas;
- Implementar classes e objetos em termos de componentes (arquivos fonte, binários, executáveis, e outros);
- Testar os componentes desenvolvidos unitariamente; e
- Integrar os resultados produzidos por desenvolvedores individuais (ou equipes) em um sistema executável.

O sistema é realizado pela implementação de componentes. O RUP descreve como reusar componentes existentes, ou implementar componentes novos com responsabilidade bem definida, tornando o sistema mais fácil de manter, e aumentando as possibilidades de reuso.

3.6.5 - Teste

Os propósitos do fluxo de trabalho de teste são:

- Verificar a interação entre objetos;
- Verificar a própria integração de todos os componentes do software;
- Verificar se todos os requisitos foram implementados corretamente; e
- Identificar os defeitos antes da implantação do software.

O RUP propõe que os testes sejam executados de forma iterativa ao longo do projeto. Isto lhe permite achar defeitos mais rapidamente, e reduz drasticamente o custo de corrigir o defeito. São feitos testes ao longo de três dimensões de qualidade confiabilidade, funcionalidade, desempenho da aplicação e desempenho de sistema. Para cada uma destas dimensões de qualidade, o processo descreve como se passa ciclo de teste: planejar, projetar, implementar, executar e avaliar.

São descritas estratégias de quando e como automatizar o teste. A automatização de teste é importante, permitindo testes de regressão ao fim de cada iteração, como também para cada versão nova do produto.

3.6.6 - Entrega

O propósito do fluxo de trabalho de entrega é de distribuir o produto gerado para seus usuários. Inclui uma gama extensiva de atividades:

- Produzir liberações externas do software;
- Empacotar o software;
- Distribuir o software;
- Instalar o software; e
- Providenciar apoio para os usuários.

Em muitos casos, isto também inclui atividades como:

- Planejar e administrar de testes iniciais;
- Migrar os dados existentes; e
- Obter o aceite formal.

Embora atividades de entrega sejam principalmente centradas na fase de Transição, muitas das atividades são incluídas em fases anteriores para preparar a entrega ao término da fase de construção.

3.6.7 - Gerenciamento do projeto

Gerenciamento do projeto de software é a arte de equilibrar objetivos, gerenciando o risco e superando limitações para entregar um produto no qual se conhece as necessidades de clientes e usuários. É fato que muitos projetos fracassam pela dificuldade desta tarefa. Este fluxo de trabalho se atenta principalmente no aspecto específico de um processo de desenvolvimento iterativo. A meta é tornar a tarefa mais fácil provendo:

- Uma estrutura para administrar projetos de software;
- Um guia prático para planejar, executar e monitorar projetos; e
- Uma estrutura para administrar risco.

Não há garantias de sucesso mas apresenta uma forma de administrar o projeto que melhorará as chances entregar software com sucesso.

3.6.8 - Gerenciamento de configuração

Neste fluxo de trabalho é descrito como controlar os numerosos artefatos produzidos pelas pessoas (papéis) que trabalham no projeto. Este controle assegura que os artefatos resultantes das iterações não estão em conflito devido a alguns dos seguintes problemas:

- Atualização Simultânea. Quando duas ou mais pessoas trabalham separadamente no mesmo artefato, a última pode fazer uma mudança e destruir o trabalho anterior.
- Notificação. Quando um problema é corrigido em artefatos compartilhados por várias pessoas, elas podem não ser notificadas da mudança.
- Versões Múltiplas. A maioria dos programas grandes é desenvolvido em versões evolutivas. Uma versão poderia estar em uso pelo de cliente, enquanto outra está em teste, e outra ainda está em

> desenvolvimento. Se forem achados problemas em qualquer uma das versões, as correções precisam ser propagadas entre eles. Por isto é necessário controlar e monitorar cuidadosamente estas mudanças para que elas possam ser propagadas quando necessário.

Este fluxo de trabalho provê diretrizes para administrar variantes múltiplas do sistemas de software que está evoluindo, localizando que versões são usadas em determinadas construções do software, executando a distribuição de programas individuais ou liberações inteiras de acordo com especificações de versão definidas pelo usuário, e obrigando políticas de desenvolvimento específicas.

É descrito como pode ser administrado o desenvolvimento paralelo, desenvolvimento feito em locais múltiplos, e como automatizar o processo de construção. Isto é importante em um processo iterativo onde pode ser necessário gerar versões diariamente, algo que ficaria impossível sem automatização.

É descrito como manter um controle das alterações indicando o motivo da alteração, quando e por quem foi alterado qualquer artefato. Este fluxo de trabalho também abrange a administração de pedido de mudança, por exemplo, como informar defeitos, administrar o seu ciclo de vida, e como usar dados de defeito para localizar progresso e tendências.

3.6.9 - Ambiente

O propósito do fluxo de trabalho de ambiente é proporcionar para a organização de desenvolvimento de software o ambiente para o desenvolvimento, processos e ferramentas são necessários para apoiar o time de desenvolvimento.

Este fluxo de trabalho se concentra nas atividades para configurar o processo no contexto do projeto. Também em atividades para desenvolver as diretrizes necessárias para apoiar o projeto. Descreve um procedimento passo a passo de como implementar o processo em uma organização.

O fluxo de trabalho de ambiente também contém um guia de desenvolvimento que proporciona as diretrizes, modelos e ferramentas necessárias para personalizar o processo. Certos aspectos do fluxo de trabalho de ambiente não estão cobertos no processo como selecionar, adquirir, criar as ferramentas de trabalho e manter o ambiente de desenvolvimento.

4 – EXTREME PROGRAMMING

4.1 – Introdução ao *Extreme Programming*

Um dos principais problemas relacionados ao desenvolvimento de software são riscos como: atrasos no cronograma, projetos cancelados devido a esses atrasos, sistemas tornando-se obsoletos, alta taxa de defeitos, mudanças de requisitos e saída de importantes membros da equipe de desenvolvimento são exemplos de risco que podem resultar no fracasso de um projeto de software [7].

Extreme Programming (XP), é um processo de desenvolvimento de software que visa alcançar duas metas almejadas pelas organizações de desenvolvimento de software:

- Desenvolvimento rápido e consistente com as reais necessidades do cliente; e
- Fácil manutenção, permitindo que o software seja modificado à medida que as necessidades do negócio se alteram ou ampliam.

Extreme programming é um conjunto bem definido de regras que vem conquistando um grande número de adeptos, particularmente entre os programadores de tecnologia orientada a objeto, em especial os desenvolvedores Java [8]. Podendo ser aplicado a projetos com altos riscos e requisitos dinâmicos.

Comunicação, Simplicidade, Realimentação e Coragem são os quatro lemas adotados pelos seguidores de XP, que correspondem a quatro dimensões nas quais os projetos podem ser melhorados. XP oferece condições para que os desenvolvedores possam responder confiavelmente a alterações de requisitos propostas pelos clientes, mesmo em estágios finais do ciclo de vida do projeto.

Entretanto, XP não se aplica bem a todo e qualquer tipo de projeto e como todo processo, possui algumas restrições. Para que sua aplicação seja produtiva são necessárias algumas características, entre elas:

- Grupos Pequenos: XP supõe que as equipes de desenvolvimento de um projeto possuem de 2 a 10 programadores;
- Trabalho em equipe: XP expande a equipe de desenvolvimento incluindo forte integração entre gerentes e clientes durante todo o processo de desenvolvimento;
- Testabilidade: É importante poder criar testes funcionais e unitários automatizados;
- Produtividade: É necessária uma equipe de desenvolvimento comprometida e dinâmica para assegurar um alto grau de produtividade, que é indispensável na realização dos projetos XP; e
- Agilidade na comunicação com o cliente: A metodologia de desenvolvimento enfoca a rapidez da implementação e, desta forma, a comunicação com o cliente deve ser ágil. É preciso que o cliente seja especialmente dedicado ao projeto e possa tomar decisões rápidas para garantir o cronograma do projeto.

Desta forma, XP requer uma mudança cultural, o que nem sempre é fácil alcançar. Além disso, alguns obstáculos à sua implementação podem surgir como: gerentes ou clientes que insistem em ter um conjunto completo de especificações ou um projeto detalhado antes da fase de codificação, ou ainda, sistemas com uma grande quantidade de aplicações já existentes e difíceis de serem alteradas não oferecem flexibilidade suficiente para garantir a simplicidade no código, um dos requisitos de XP.

4.2 - Ciclo de vida e as fases do processo XP

O ciclo de vida XP é uma das abordagens mais discutidas por diversos grupos que adotam este processo. A figura 6 representa o ciclo de vida XP.

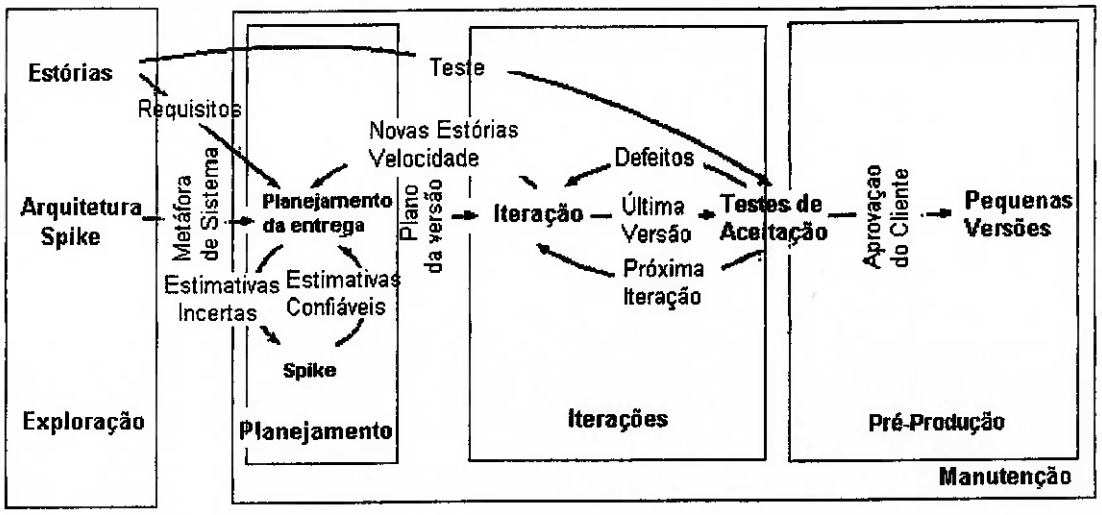

Figura 6 – Ciclo de vida do XP.

O ciclo de vida XP é bastante curto e, à primeira vista, difere dos padrões dos modelos de processo convencionais. No entanto, esta abordagem pode fazer sentido em um ambiente onde as mudanças de requisitos do sistema são fatores dominantes. Na fase de planejamento os requisitos do cliente são cuidadosamente coletados à medida que são fornecidos. A seguir, os testes são elaborados a partir das especificações do cliente, e a fase de codificação é realizada visando atender esses testes.

Existe uma relação próxima e contínua entre as fases de teste e codificação. E, por fim, o sistema é novamente projetado (ou reconstruído) à medida que novas funcionalidades são incorporadas.

A seguir descrevem-se as fases [7].

4.2.1 - Exploração

A Exploração é uma fase inicial onde o projeto ainda não está em produção. Nesta fase é necessário que os membros da equipe tenham se certificado que possuem as ferramentas corretas e que possuem (ou conseguem aprender) as habilidades necessárias para iniciar o projeto.

Durante a fase de Exploração os programadores fazem testes para estabelecer a arquitetura do sistema. Eles exploram as possibilidades testando vários tipos de arquitetura possíveis. Se possível deve-se simular cargas realistas para validar as arquiteturas criadas. Esta exploração arquitetônica é muito importante quando o usuário solicitar a implementação de estórias que a equipe não sabe como implementar. Enquanto a equipe estiver criando e testando a arquitetura o cliente escreve as estórias.

A fase de Exploração é terminada quando o cliente está confiante que as estórias foram suficientemente levantadas e os programadores estão confiantes que podem estimar o que deve ser implementado.

4.2.2 - Planejamento

O propósito da fase de Planejamento é que os clientes e os membros da equipe entrem em acordo em qual data e qual o conjunto mais importante de estórias será entregue na primeira versão. A forma como isto é feito será abordada no item atividade de planejamento.

4.2.3 - Iterações

Nesta fase o compromisso acordado na fase anterior é dividido em iterações de quatro semanas. Cada iteração produzirá um conjunto de casos de teste funcionais para as estórias que serão realizadas.

A primeira iteração criará a arquitetura e para esta iteração devem ser escolhidas as estórias que forçam a criar o sistema inteiro mesmo que em forma de esqueleto. Para as iterações subseqüentes o cliente deve indicar as estórias mais importantes.

Idealmente, ao término de cada iteração o cliente terá executado os testes funcionais e validado a versão.

4.2.4 - Produção

O término de uma versão aumenta o ciclo de avaliação e as iterações passam a ser de uma semana. Tipicamente haverá algum processo para certificar que o software está pronto para entrar em produção. Neste momento será verificado se é necessário ajustar o desempenho do sistema.

Durante esta fase, será reduzida a velocidade de evolução do sistema. Isto não indica que ele não evoluirá mas a avaliação se uma mudança pode ser feita ou não deverá ser feita com mais cautela.
Esta fase termina quando o sistema está pronto para entrar em produção.

4.2.5 - Manutenção

Manutenção é o estado normal de um projeto XP. Nesta fase deve-se produzir novas funcionalidades, manter o sistema existentes e alterar a equipe caso necessário. Portanto as fases Planejamento Iteração e Produção são revisitadas.

Toda entrega de uma versão começa com a fase de exploração. Onde poderá ser feitos grandes reestruturações e reconstruções no sistema, testar novas tecnologias ou testar novas idéias arquitetônicas.

Desenvolver um sistema que está em produção não é igual a desenvolver um sistema que não está ainda em produção. Deve-se ter mais cuidado ao executar as mudanças. Deve estar preparado para parar o desenvolvimento novo e atender um problema da versão que está em produção.

É provável que estando em produção a velocidade de desenvolvimento mude e as novas estimativas sejam mais conservadoras.

4.3 - Atividades

Durante o ciclo de vida de um projeto XP são aplicadas as 4 atividades: Planejamento, Teste, Codificação e Projeto em cada atividade são executadas algumas tarefas que serão detalhadas a seguir[7].

4.3.1 – Planejamento

Planejamento consiste em estimar diversos fatores que podem afetar o desenvolvimento do software. Algumas das tarefas do planejamento incluem: decidir escopo e prioridade do projeto, estimar custos e cronogramas e criação de um plano para a entrega de uma nova versão do produto. Uma diferença, entre o XP e a maioria dos modelos de processo convencionais, é que XP não define a especificação formal e completa de requisitos.

Estórias de usuário

"Estórias de usuário" [10] são semelhantes a casos de uso e têm a finalidade de auxiliar a elaboração de estimativas de tempo para o planejamento do software a ser entregue. Também podem ser uma alternativa para declarações de requisitos formais. Estórias de usuário são escritas pelos usuários, utilizando uma linguagem natural, ao invés de um vocabulário técnico, para especificar as tarefas que o sistema precisa realizar. Trata-se de um processo semelhante a cenários de caso uso, mas não se limita a descrever a interface do programa. As estórias sempre focam a necessidade do usuário e, por serem escritas em uma linguagem natural, permitem a compreensão independentemente de uma tecnologia específica, projeto de dados ou algoritmo.

As estórias de usuário são também utilizadas para a criação de testes de aceitação que verificam se o requisito foi corretamente implementado. A única diferença entre as estórias de usuário e o documento de requisitos é que o último apresenta maior grau de detalhe. As estórias de usuário devem possuir detalhes suficientes para estimar, com baixo risco, quanto tempo levará a implementação. Na atividade de implementação durante a fase de Iteração os requisitos serão mais bem explicitados através da interação dos desenvolvedores com os clientes.

Para cada estórias de usuário, a equipe de desenvolvedores estima inicialmente qual seria o tempo ideal para o desenvolvimento. O tempo ideal é o tempo necessário para codificar a estória supondo que não existam outras tarefas a serem feitas e os desenvolvedores saibam exatamente como codificar o problema.

Se o tempo ideal resultante for menor que uma semana essa estória é considerada como um detalhe, e deve ser combinada à outra estória de escopo mais amplo. Se o tempo ideal for maior do que três semanas, deve-se analisar a possibilidade de subdividir a estória em tarefas mais específicas. Um plano para a entrega de um produto deve conter entre 60 e 100 estórias.

A tarefa de levantamento das estórias de usuário é mais executada na fase de Exploração e depois revisitada durante a fase de Iteração e sempre que uma nova alteração do sistema é necessária [7].

Planejamento da entrega

A tarefa de planejamento da entrega é executada na fase de planejamento. Esta tarefa consiste em reuniões para planejamento da entrega de uma nova versão do produto e define as características gerais do software. O plano de entrega é utilizado para criar o planejamento das iterações de desenvolvimento das versões do produto.

O processo de desenvolvimento XP é incremental e a cada iteração do software um conjunto de estórias é implementado. Durante a reunião para criação do plano de entrega do software, um conjunto de estórias semelhante é agrupado e determina-se o que será feito em cada iteração do ciclo através da utilização de cartões com as estórias identificadas. Estes cartões são organizados em ordem de implementação para as próximas versões do software. O cronograma para a entrega de uma nova versão do software é baseado no escopo das estórias definidas para a implementação no ciclo de iteração.

Os ciclos de desenvolvimento são curtos com entregas freqüentes de novas versões do software. O plano de entrega permite que as decisões sejam tomadas de forma a assegurar a viabilidade técnica e comercial do projeto. As regras para a criação do plano de entrega envolvem métodos de negociação de cronograma, que permitem o comprometimento das áreas comercial e técnica da empresa.

A única tarefa extra, além da codificação existente no tempo ótimo de programação, é a seção de testes. Se as estimativas para implementação das estórias não agradarem a gerência ou a área comercial, ao invés de subestimar o tempo de implementação, deve-se diminuir o escopo das estórias que serão entregues na próxima versão do software.

Pode-se definir a velocidade do projeto por tempo ou por escopo. A estimativa por tempo considera o número de estórias que podem ser implementadas em um dado período, enquanto a estimativa por escopo analisa o tempo necessário para implementação de um dado conjunto de estórias. Ambas as formas de estimativa consideram os recursos humanos disponíveis que podem ser alocados de acordo com as restrições de tempo e escopo da versão.

A gerência ou área comercial da empresa pode definir apenas 2 das 3 variáveis de projeto: tempo, escopo e recursos humanos. A variável restante sempre será ditada pela equipe de desenvolvimento de forma a viabilizar a entrega da próxima versão.

Métricas

Duas formas para estimar e medir o progresso de um projeto são o Fator de Carga e Velocidade do projeto. A velocidade do projeto é utilizada para monitorar e estimar um projeto individual. Contudo a velocidade não é proveitosa para comparar novos projetos ou novas estimativas. Alem disso, a velocidade trabalha melhor quando a equipe de desenvolvimento é estável e o tempo de desenvolvimento nas iterações são constantes. A média do Fator de Carga de muitos projetos são mais úteis para um plano inicial do projeto e para projetos pouco estáveis.

O Fator de carga é calculado pela diferença entre o tempo ideal estimado e o tempo decorrido dividido pela estimativa ideal. Multiplica-se a estimativa pela média do fator de carga para estimar o tempo atual de desenvolvimento. O fator de carga varia consideravelmente entre as pessoas, projetos e ambientes de desenvolvimento devido ao número de perturbações do ambiente, como por exemplo reuniões e outros

fatores. Tipicamente o fator de carga tende a ser uma faixa entre dois e meio a três semanas do real esforço para cada semana estimada ideal. Assim sendo:

- Fator de carga = (tempo real estimado–gasto)/tempo Ideal Estimado
- Ajuste de tempo Estimado = Tempo Ideal Estimado * Média(Fator Carga)

A Velocidade do projeto é uma forma de medir o ritmo do projeto com base no número de estórias ou tarefas completadas pelo total estimado. A velocidade é usada para prever durante o planejamento de uma nova versão se o escopo se ajusta ao tempo alocado.

Velocidade = Número de estórias desenvolvidas/ Total Estimado

As métrica são utilizadas durante a fase de Planejamento para auxiliar o planejamento da entrega e durante a Iteração para o acompanhamento da velocidade do projeto.

Planejamento das iterações de desenvolvimento

Esta tarefa é realizada durante o início da fase de Iteração e consiste em reuniões de planejamento. Nessa reunião, são selecionadas as estórias mais importantes (definidas pelo cliente) para a versão, além dos testes de aceitação que apresentaram problemas no ciclo anterior.

As estórias são divididas em tarefas de programação que devem ter tempo ideal de codificação de 1 a 3 dias. Tarefas menores que 1 dia podem ser agrupadas e maiores que 3 dias podem ser quebradas. O fator de velocidade do projeto é utilizado para verificar se a quantidade de tarefas condiz com o tempo estimado para aquela iteração. Caso se detecte uma sobrecarga na quantidade de tarefas, o cliente é contatado para definição das tarefas que serão adiadas para a próxima iteração. Da mesma forma, caso se detecte superdimensionamento do tempo, novas estórias podem ser aceitas para o ciclo de iteração.

Reuniões rápidas

As reuniões semanais de toda a equipe de desenvolvimento para discussões gerais sobre o projeto geralmente são consideradas uma tarefa complexa que consome muito tempo e muitas vezes se mostra improdutiva. Uma alternativa proposta por XP são as reuniões rápidas. Estas são realizadas todos os dias, em um dado horário, para discutir problemas e soluções, além de orientar a equipe para o que deve ser feito.

Essa abordagem evita a complexidade de estabelecer um horário durante um dia da semana no qual toda a equipe de desenvolvimento possa comparecer.

Essas reuniões rápidas de verificação da situação do projeto contribuem para o conhecimento geral do projeto por todos os membros da equipe. Problemas específicos podem ser discutidos em reuniões separadas, onde compareçam apenas as pessoas envolvidas com o problema em questão. Tais reuniões podem ter caráter prático e informal com o líder de projeto e alguns desenvolvedores, revisando certo código em frente a um computador ou projetando soluções em papel ou em uma lousa.

Estas reuniões são executadas em todas as fases do projeto. Porém são mais utilizadas nas fases de Planejamento, Iteração e Manutenção.

4.3.2 - Teste

Um dos requisitos básicos de XP, é o de escrever testes antes do código. Os testes em XP são divididos em duas categorias: testes unitários e testes de aceitação (também chamados de testes funcionais) [11]. Testes unitários são, em geral, escritos pelos desenvolvedores e têm a finalidade de testar uma classe individual ou um pequeno grupo de classes. Já os testes de aceitação são usualmente escritos pelos próprios clientes ou por uma equipe de teste externa (com a ajuda dos desenvolvedores) e têm a finalidade de testar todo o sistema.

As atividades de teste são realizadas durante todo o processo de desenvolvimento sendo mais fortemente executados durante a fase de Iteração. Todo código é construído com o propósito de satisfazer os resultados esperados. E, à medida que

um novo código é adicionado, novos testes devem ser realizados para assegurar que não ocorram impactos negativos.

Testes Unitários

Os testes unitários são um dos elementos chave em XP, pois são criados antes do código e armazenados em um repositório junto ao código que será testado. Um código que não possua seu respectivo teste unitário não deve ser liberado, pois cada parte do sistema com possibilidade de falha precisa ser verificada. Além disso, a criação de testes unitários antes do código provê uma melhor compreensão dos requisitos e direciona o foco dos desenvolvedores.

Um fator importante para a utilização de testes unitários, especialmente se estes forem automatizados, é a economia de custo que podem proporcionar ao identificar e oferecer proteção contra erros. É relevante considerar que descobrir todos os problemas que podem ocorrer durante o desenvolvimento consome uma grande quantidade de tempo, tornando-se necessário que um conjunto completo de testes unitários esteja disponível logo no início, e não no último mês do projeto.

Nesta atividade os desenvolvedores criam e executam testes unitários toda vez que um código é escrito ou modificado. Consertar pequenos problemas assim que são encontrados, em geral, leva menos tempo do que consertar grandes problemas a poucas horas do prazo final. Outro benefício dos testes unitários automatizados é a possibilidade de combinar um conjunto de alterações com a última versão liberada e liberar novas versões em um curto espaço de tempo.

Testes de Aceitação

Os testes de aceitação constituem uma das principais diferenças entre o XP e os processos de desenvolvimento tradicionais. Em geral, são escritos pelos clientes ou usuários finais através das estórias de usuário [7], com a assistência de um indivíduo da equipe responsável por testar o software. Durante uma iteração, as estórias de usuário selecionadas durante a reunião de planejamento de iteração serão traduzidas em testes de aceitação. Uma estória de usuário pode ter um ou mais testes de

aceitação para assegurar que a sua funcionalidade esteja de acordo com a especificação. Após a implementação da estória de usuário, o cliente especifica cenários para serem testados, e ela não é considerada completa até que tenha passado por esses testes de aceitação. Isso significa que novos testes de aceitação devem ser criados a cada iteração, ou a equipe de desenvolvimento não deverá reportar progresso.

Testes de aceitação são testes de sistema de caixa preta e cada um deles representa algum resultado esperado. Testes de aceitação também são usados como testes de regressão anteriores à liberação de uma versão do software.

Nesse contexto, a garantia de qualidade é uma parte essencial do processo XP. Em alguns projetos a garantia de qualidade é realizada por um grupo especializado, enquanto em outros a garantia de qualidade é integrada à própria equipe de desenvolvimento. Em ambos os casos o XP requer que o desenvolvimento tenha uma relação muito próxima com as atividades de garantia de qualidade.

Dentre as principais metas dos testes de aceitação estão: fornecer aos clientes, gerentes e desenvolvedores a confiança de que o produto inteiro está progredindo na direção certa; e checar cada incremento no ciclo XP para verificar se o valor de negócio está presente.

Os testes de aceitação, que são responsabilidade do testador e do cliente, são testes de todo o sistema sob a perspectiva do cliente, e não testes que verificam cada caminho possível no código (essa é a finalidade dos testes unitários).

4.3.3 - Codificação

Essa atividade do processo está focada nos métodos para a codificação dos módulos que compõem o projeto. Em projetos de XP, a qualidade do código é muito importante. Para tentar manter esta qualidade, algumas práticas são utilizadas, como: programação aos pares, integração contínua e a criação dos testes antes da criação do

código. Além disso, a participação contínua do cliente é indispensável para que a equipe de desenvolvimento faça um bom trabalho.

Uma outra forma de manter a boa qualidade da codificação é usar padrões de codificação definidos. Além disso, as equipes de desenvolvimento não podem fazer horas extras para terminar o projeto dentro do prazo, pois os estudiosos da área acreditam que horas extras apenas diminuem a motivação dos desenvolvedores e, conseqüentemente, o rendimento e a qualidade do trabalho realizado.

As atividades de codificação são fortemente exploradas em todas as fases do processo já que o código é o produto mais importante de um projeto XP.

Programação aos Pares

Esta pode ser considerada uma das abordagens mais controversas de XP. Todo código será produzido por duas pessoas trabalhando juntas em um único computador. O objetivo da programação aos pares é reduzir as chances de se produzir um código ruim, encorajar o espírito de colaboração entre os programadores, diminuir os riscos gerados quando uma única pessoa tem o conhecimento de partes do código, assegurando que todos os elementos da equipe de desenvolvimento tenham a visão completa do projeto.

Assim, se alguma tarefa exigir mais esforço, será fácil alocar qualquer outro desenvolvedor sem a necessidade de que este tenha que aprender a partir do ponto inicial. Se apenas uma pessoa (normalmente o líder técnico ou gerente) tem a visão global do projeto, o projeto fica preso à disponibilidade de tempo e aos conhecimentos dessa pessoa. Se esta decidir abandonar o projeto as conseqüências podem ser avassaladoras.

A programação aos pares, aliada ao escalonamento da equipe de desenvolvimento, permite a disseminação do conhecimento técnico e do projeto.

Assim, permite-se a flexibilização das equipes contribuindo para maior produtividade e uma melhor distribuição da carga de trabalho.

Estratégia de Integração Contínua

Em muitas equipes de desenvolvimento, ao terminar a codificação de um novo módulo de código, os desenvolvedores fazem alguns testes e já o integram ao projeto. Como essa integração pode ser feita de forma paralela, códigos que nunca foram testados juntos acabam sendo combinados, causando assim numerosos problemas.

Um problema semelhante pode acontecer com os testes unitários. Se os desenvolvedores não tiverem acesso a um conjunto completo, correto e consistente de testes unitários, pode-se acabar procurando por erros que não existem e ignorar erros que realmente existem e não estão sendo considerados.

Algumas opções de solução para os problemas gerados pela integração paralela são propostas a seguir.

A primeira opção é que os desenvolvedores de cada módulo assegurem a correta integração e revisão do mesmo. Isto reduz o problema de erros, mas se houver forte dependência entre os módulos, o problema persiste.

Outra opção é apontar um integrador ou uma equipe de integração. Dois problemas surgem com esta opção: como adequar a quantidade de pessoas da equipe de integração a cada um dos projetos; e quantas vezes por semana ou por mês a integração deverá ser feita a fim de que os desenvolvedores não trabalhem com módulos obsoletos do projeto.

A solução mais simples geralmente adotada é a integração estritamente seqüencial feita pelos próprios desenvolvedores. Essa opção exige que os códigos sejam armazenados em repositórios protegidos e um sistema de travas seja usado pelos desenvolvedores, mantendo assim a integridade das informações do repositório após

alterações. Para que isso seja possível, uma integração contínua dos módulos é necessária; pois assim evita-se ou identifica-se facilmente os problemas de compatibilidade nas diferentes versões geradas pelas várias equipes depois da inclusão ou modificação de novos módulos.

A integração é o tipo de atividade que deve ser feita para que não se tenha mais trabalho no futuro, ou seja, se os desenvolvedores integrarem gradativamente pequenas partes novas ao projeto, será mais fácil e rápido a integração do projeto todo. Caso contrário o projeto acabará gastando muito tempo depois do seu término para realizar a integração, podendo ocasionar atraso na entrega do projeto.

Código coletivo

O código coletivo encoraja todos a contribuir com novas idéias em todas as partes do projeto. Qualquer desenvolvedor pode mudar qualquer parte do código para adicionar novas funcionalidades, concertar erros, ou refazê-lo. Contudo, para que esse tipo de trabalho funcione é necessário um controle rígido sobre quem executa as mudanças e quando essas mudanças são feitas, garantindo assim a integridade do código que está armazenado no repositório.

Isto pode ser feito exigindo-se que, quando um dos desenvolvedores alterar um determinado módulo, testes unitários para esse módulo sejam criados enquanto as alterações estiverem sendo feitas. Após o término das alterações no módulo, todos os testes unitários são verificados para se ter certeza de que o código está de acordo. Se isso acontecer, ele pode ser liberado e colocado no repositório.

É ainda necessário manter o histórico de tudo o que foi feito por cada um dos desenvolvedores, incluindo: testes unitários, códigos adicionados, erros encontrados e funcionalidades que foram alteradas em cada módulo.

O planejamento da iteração de desenvolvimento enfatiza o escalonamento de pessoas na divisão das tarefas de programação.

4.3.4 - Projeto

Em XP [10], o projeto é responsabilidade de toda a equipe e não de apenas uma pessoa. Desta forma, todos os membros da equipe podem cooperar para a elaboração de um projeto com melhor resultado do que o melhor dos projetistas poderia produzir individualmente.

Um dos aspectos mais polêmicos de XP é sua rejeição à elaboração de um projeto antecipado em prol de uma abordagem mais evolucionária. Esse aspecto é considerado por seus opositores como um retorno ao desenvolvimento baseado em tentativa e erro. Porém seus adeptos acreditam que se o código for bem elaborado, um bom projeto surgirá como conseqüência. Além disso, XP dispõe de práticas como integração contínua, teste e reconstrução, que tornam o projeto incremental possível.

Simplicidade

Um dos principais princípos de XP é: sempre faça o mais simples possível. A simplicidade deve ser mantida durante o maior tempo possível. Essa característica pode ser entendida se considerarmos o fato que um projeto simples sempre leva menos tempo para acabar do que um projeto complexo e também é muito mais fácil de entender e modificar. No entanto, é importante ressaltar que manter a simplicidade de um projeto é um trabalho difícil.

Além disso, é preciso definir o que é simplicidade. Caracterizar um sistema simples como aquele que passa em todos os testes, garante clareza de código, não possui duplicação e usa o menor número de classes e métodos possível, é uma alternativa. Por outro lado, a restrição de simplicidade de um projeto não precisa ser tão rígida, afinal, a reconstrução deve e será feita mais tarde.

A busca da simplicidade deve ser feita em todas as fases do ciclo de vida de um projeto XP.

Metáfora do sistema

A metáfora do sistema é criada na fase de Exploração do projeto. Uma metáfora do sistema é o que XP utiliza, no lugar de uma arquitetura formal, para descrever como o sistema funciona. Tipicamente, envolve um conjunto de classes e padrões que formam o núcleo do sistema em construção.

A metáfora ajuda as pessoas envolvidas no projeto a compreender o sistema sem a necessidade de um conhecimento específico, na maioria das vezes, difícil de adquirir. Para isso, o sistema é descrito em poucos parágrafos, deixando de lado o uso de termos técnicos, facilitando a compreensão não só pelos desenvolvedores, mas também pelos clientes.

Cartões CRC

Os cartões CRC são utilizados para gravar as responsabilidades e colaboradores das classes, por isso o nome derivado de Classe Responsabilidade Colaboração, estes cartões são mais utilizados durante a fase de Iteração do projeto XP

Individualmente, os cartões CRC são usados para representar objetos. A classe do objeto pode ser escrita no topo do cartão, as responsabilidades listadas do lado esquerdo e as classes de colaboração listadas à direita de cada responsabilidade. Esses cartões são utilizados em uma sessão CRC para simular o sistema e, passo a passo, superar as fraquezas e os problemas do processo.

Um dos maiores benefícios dos cartões CRC é o estímulo à criatividade; e uma das maiores críticas é a falta de projeto escrito o que, se necessário, pode ser resolvido com a retenção de alguns cartões CRC, completamente preenchidos, para documentação.

Soluções específicas

Uma solução pontual pode ser criada concentrando-se apenas no problema que está sendo examinado e ignorando-se todas as outras preocupações. Desta forma, soluções potenciais poderão ser exploradas.

O objetivo dessa abordagem é reduzir o risco de um problema técnico ou aumentar a confiabilidade das estimativas.

Soluções específicas podem ser mais necessárias durante a fase de Exploração quanto os programadores fazem testes para estabelecer a arquitetura do sistema e na fase de Produção quando otimizações podem ser necessárias.

Avaliação da necessidade

O projeto, como dito anteriormente, é elaborado em partes e, desta forma, cada ciclo de iteração busca adicionar somente a funcionalidade especificada pelas estórias de usuário escolhidas. XP considera que grande parte das tarefas extras não chegarão a ser utilizadas e apenas o que é necessário para hoje deve ser levado em consideração. As funcionalidades extras aparecerão naturalmente quando forem necessárias.

O objetivo desta prática é diminuir a complexidade do projeto ao máximo e facilitar o teste, a alteração e a reconstrução. Também contribui para a agilidade do processo de iteração garantindo o cumprimento do cronograma de entrega da versão.

Reconstrução

A reconstrução baseia-se na remoção de redundância, eliminação de funcionalidades inúteis, e reconstrução de projetos obsoletos. Essa prática garante a atualização constante do projeto em XP.

Além disso, a reconstrução praticada durante todo o ciclo de vida do projeto, traz alguns benefícios como: economia de tempo e aumento da qualidade; manutenção da simplicidade do projeto à medida que esse evolui; a desordem e a complexidade desnecessária são evitadas; e o código permanece claro e conciso garantindo maior facilidade de compreensão, modificação e extensão.

4.4 - As quatro dimensões do XP

XP segue um conjunto de dimensões, princípios e requisitos básicos que visam alcançar eficiência e efetividade no desenvolvimento de software [7]. As dimensões são quatro: Comunicação, Simplicidade, Realimentação e Coragem.

4.4.1 - Comunicação

XP leva em consideração a grande importância da comunicação, mas também permite não fazer nenhuma documentação, ou muito pouca. Talvez seja justamente por isso que XP tem conquistado jovens programadores.

A comunicação é definida como um fator chave para o sucesso de qualquer projeto, seja ele usando XP ou outra metodologia.

Várias práticas de XP promovem uma maior comunicação entre os membros da equipe. A comunicação não é limitada por procedimentos formais. XP indica que se use o melhor meio possível, que pode ser:

- Uma conversa ou reunião informal;
- Um e-mail, um telefonema;
- Diagramas, se necessário (pode, mas não precisa, ser UML);
- O próprio código;
- Estórias elaboradas pelo usuário-final; e
- etc...

Também deve se dar preferência à comunicação mais ágil:

- Telefonema no lugar de e-mail;
- Presença física no lugar de comunicação remota; e
- Código auto-explicativo no lugar de documentação escrita.

4.4.2 - Simplicidade

XP preconiza que é melhor fazer algo simples hoje e pagar um pouco mais amanhã se mudanças forem necessárias, do que fazer algo mais complicado hoje e não usá-lo amanhã. Por exemplo:

- Deve se projetar somente aquilo que está contido nos requisitos;
- Deve-se programar somente aquilo que está no projeto; e
- Deve-se testar somente aquilo que está contido nos requisitos.

O resultado é que em XP termina-se de escrever um código sabendo que mais tarde será reescrito.

4.4.3 – Realimentação

É uma forma de determinar o corrente estágio do sistema.

Também envolve liberar versões mais rápido, isto é, ter uma iteração do projeto com o cliente tão logo quanto possível, assim poderá perceber o que o cliente esta achando do produto.

Isto promove um valoroso retorno do cliente para o programador.

Várias práticas do XP garantem uma rápida realimentaçao sobre várias etapas do processo:

- Realimentação sobre qualidade do código (testes unitários, programação em pares, propriedade coletiva).
- Realimentação sobre estado do desenvolvimento (estórias do usuário-final, integração contínua e planejamento).

Por XP permitir uma maior agilidade:

- Erros detectados são corrigidos imediatamente;
- Requisitos e prazos reavaliados mais cedo;
- Facilita a tomada de decisões;
- Permite estimativas mais precisas; e
- Maior segurança e menos riscos para investidores.

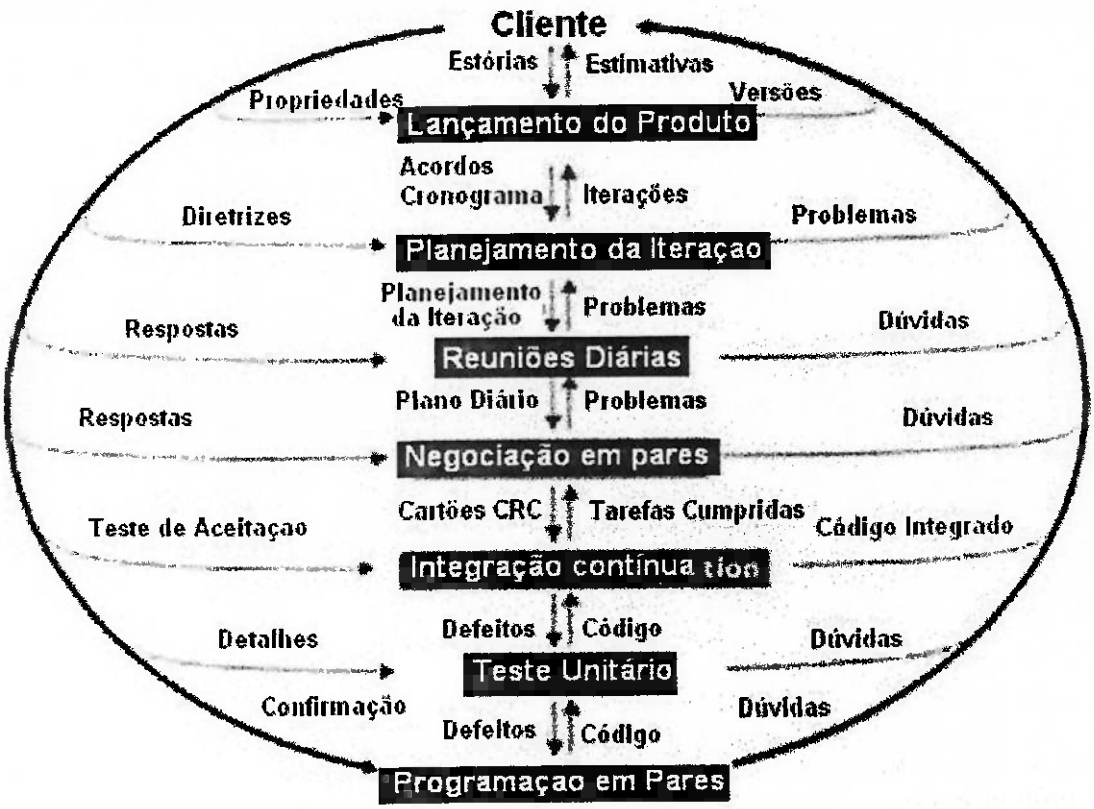

Figura 5 – Ciclo de realimentação do XP.

4.4.4 - Coragem

A coragem está relacionada a enfrentar o risco de fazer alterações e propor algo novo e melhor que simplifique o desenvolvimento. XP incentiva os programadores a terem coragem para correr este risco.

Testes, integração contínua, programação em pares são algumas das práticas do XP que aumentam a confiança do programador e o ajudam a ter coragem para:

- Melhorar o desenho do código que está funcionando para torná-lo mais simples;
- Reestruturar o código desnecessário;
- Investir tempo no desenvolvimento de testes;
- Alterar o desenho da arquitetura do projeto em estágio avançado do projeto;
- Pedir ajuda aos que sabem mais;

- Dizer ao cliente que um requisito não vai ser implementado no prazo prometido; e
- Abandonar processos formais e fazer desenho da arquitetura e documentação em forma de código.

4.5 - Princípios básicos

As dimensões: Comunicação, Simplicidade, Realimentação e Coragem são muito vagas para nos ajudar a decidir quais práticas usar. Então, as dimensões são fundamentadas em alguns princípios mais concretos para serem utilizados por XP:

1) Planejamento, também chamado de planejar o jogo: O processo de planejamento de XP permite que se defina o valor de negócio dos recursos desejados e utilize estimativas de custo fornecidas pelos programadores para decidir o que é necessário ser feito e o que pode ser adiado.

2) Pequenos lançamentos: As equipes XP colocam um sistema simples em produção com antecedência e o atualizam freqüentemente em ciclos bastante curtos.

3) Metáforas do sistema: As equipes XP utilizam um sistema de nomes e uma descrição do sistema sem a utilização de termos técnicos, para guiar o desenvolvimento e a comunicação como cliente.

4) Projeto simples: Um programa construído através do método XP deve ser o mais simples possível satisfazendo os atuais requisitos, sem a preocupação de atender outros que surgirão no futuro. O foco está em prover valor de negócio.

5) Teste: As equipes XP focalizam a validação do software durante todo o processo. Os programadores desenvolvem software escrevendo primeiro os testes, e só então o software que atenda aos requisitos desses testes. Os clientes provêm testes de aceitação para ter certeza que os recursos necessários estão sendo fornecidos.

6) Reconstrução: As equipes XP procuram aperfeiçoar o projeto do sistema durante todo o desenvolvimento, mantendo a clareza do software: sem ambigüidade, com alta comunicação, simples, porém completo.

7) Programação aos pares: Os programadores XP produzem o código em pares, ou seja, dois programadores trabalhando juntos na mesma máquina. Muitos experimentos têm mostrado que a programação aos pares produz software de melhor

qualidade com um custo similar ou menor do que o produzido por programadores trabalhando individualmente.

8) Propriedade coletiva: Todo o código pertence a todos os programadores. Essa característica permite que a equipe trabalhe a toda velocidade, uma vez que as alterações podem ser feitas sem atrasos, pois todos têm liberdade para fazê-las.

9) Integração contínua: As equipes XP integram e constróem o sistema de software várias vezes por dia. Isso mantém todos os programadores em sintonia e possibilita um progresso rápido.

10) Quarenta(40) horas de trabalho semanal: Programadores exaustos cometem mais erros. As equipes XP não trabalham por um tempo excessivo, mantendo-se, assim, mais efetivas.

11) Cliente dedicado: Um projeto XP é conduzido por um indivíduo dedicado (um cliente), que determina os requisitos, atribui as prioridades, e responde as dúvidas dos programadores (relacionadas aos requisitos). Essa prática melhora a comunicação e gera menos documentos, o que, em geral, é uma das partes mais caras num projeto de software.

12) Código padrão: Para que uma equipe trabalhe em pares de forma efetiva e compartilhe a propriedade de todo o código, todos os programadores precisam escrever da mesma forma, com regras que assegurem a clareza do código.

5 – COMPARAÇÃO ENTRE RUP E XP

Conforme vistos nos capítulos anteriores XP e RUP são exemplos de processos de desenvolvimento de software, ambos fluem de forma iterativa. O RUP é um processo mais robusto que pode se adaptar a vários tipos de projetos podendo ser considerado tanto como uma metodologia tradicional ("pesada") como uma metodologia ágil ("leve").

RUP e XP compartilham características de modo que cada um utiliza as melhores práticas de desenvolvimento de software, mas os dois métodos diferem em termos da sua adaptabilidade.

5.1 - Componentes de XP e RUP

A maioria dos processos tem alguns elementos comuns que tornam uma comparação sistemática possível. Eles requerem para seqüências ou grupos de atividades que são executadas através de papéis para gerar artefatos ou produtos de trabalho, onde alguns ou todos são entregues ao cliente. Assim uma base para comparar as duas metodologias em termos de componentes seria [15]:

- Tempo e distribuição de esforço - como cada processo é organizado com o passar do tempo;
- Artefatos – quais os produtos de trabalho são gerados, e produzidas ao longo de um projeto baseado em XP ou RUP;
- Atividades - métodos específicos de produzir cada artefato aplicável dentro do processo;
- Disciplinas - comparação de modo no qual XP e RUP delineiam as áreas principais de preocupação em engenharia de software; e
- Papéis - Diferenças entre papéis em RUP e papéis (posições) dentro de uma equipe em XP.

5.2 – Uma Breve Comparação

Em relação ao XP nota-se a maioria dos seus princípios de XP apresentam práticas de bom senso que fazem parte de qualquer processo[16]. Isto se origina do fato que

nem sempre se pretendia que XP se tornasse um processo ou uma metodologia tradicional, mas um processo leve onde se deveria construir casos de teste até mesmo antes do começo da construção.

A relação entre tempo e distribuição de esforço e as atividades são perfeitos para um paradigma de desenvolvimento onde o teste é o foco. Considerando que as disciplinas planejadas são discretamente nomeadas, o número de papéis também é destinado para um grupo de desenvolvimento pequeno. O fato de XP procurar ser simples nas atividades, e o fato de procurar dar foco as atividades que são realmente de alta prioridade, as partes mais valiosas do sistema e que são identificadas no momento. Em lugar de soluções artificiosas para problemas que não são tão importantes fazem com que XP seja considerada uma metodologia importante.

Há lados potencialmente ruins, um deles é a dificuldade de incorporar programação em pares sem ter algum treinamento nesta metodologia. Também, projetar em pequenas iterações não é natural a todos os membros da equipe, e a maioria deles na verdade trabalha com a meta especulativa até o final, e se familiarizam muito tarde com a metodologia no desenvolvimento[17].

Já o RUP, observa-se que foi criado para estabelecer uma estrutura de processo global. É planejado para ser geral e amplo o bastante para ser usado por todos os tipos de organizações de desenvolvimento de software, especialmente nas que faltam uma cultura de processo forte.

Isto é substanciado pelo fato que existe um número grande de artefatos e papéis definidos dentro do processo de desenvolvimento, e estes podem ser adaptados a cada tipo de organização.

Olhando em detalhes para todos os componentes dos dois processos, observa-se que RUP precisa ser configurado. Esta flexibilidade fundamental permite que ele seja adequado a vários tipos de projeto.

XP e RUP dão ênfase a desenvolvimento de software de iterativo, procurando impedir que recursos fiquem inativos durante o processo de desenvolvimento. Eles diferem na freqüência de iterações. XP executa mais atividades de desenvolvimento em qualquer iteração, fazendo com isto que seja mais fácil de administrar mudanças e identificar corrigir defeitos.

O XP é falho ao estabelecer a arquitetura dos componentes para o processo de desenvolvimento. Não especifica como dividir o sistema em componentes exatamente, e não sugere como fazer uso de componentes existentes. A falta de uma arquitetura definida em XP é um de seus elementos mais criticados. Por outro lado alguns críticos acreditam que a falta de uma arquitetura completa acrescenta à habilidade de XP para adaptar requisitos variáveis.

Tanto XP como RUP especificam o rastreamento das mudanças, mas diferem na forma de execução. Em um projeto XP, mudanças podem ser feitas por qualquer um a qualquer parte do sistema. Quando são feitas mudanças, elas são anotadas para notificar aos outros. No RUP porém, vai-se mais adiante. Em qualquer tempo uma mudança é feita, então a arquitetura, modelo visual, e qualquer outra documentação também são atualizadas. Este rastreamento de mudança cuidadoso assegura que todos os membros da equipe estão atentos a uma mudança, mas também requer uma maior quantia de esforço e de recurso para acompanhar cada mudança.

5.3 - Os doze princípios

Os processos de software possuem elementos comuns como fluxo de trabalho, artefatos gerados, definição de papéis para os envolvidos e práticas de trabalho. O XP e o RUP diferem muito quanto a definição do fluxo de trabalho, artefatos gerados e definição dos papéis em virtude do RUP ser um processo mais amplo e criado para ser adaptável a vários tipos de projetos e organizações. Os doze princípios básicos apontados em XP serão utilizados neste estudo comparativo. São eles:

- Planejamento;
- Pequenos lançamentos;
- Metáforas do Sistema;

- Projeto Simples;
- Teste;
- Reconstrução;
- Programação aos pares;
- Propriedade coletiva;
- Integração contínua;
- Quarenta (40) horas de trabalho semanal;
- Cliente dedicado;e
- Código padrão.

5.3.1 – Planejamento

Quando se trata de planejamento, RUP e XP concordam: planos mudam, e não é possível planejar um projeto completo em detalhes. O melhor a fazer é se antecipar às mudanças e assegurar que os riscos associados estão sob controle. De acordo com XP, deveria-se priorizar as estórias que se tem que atender no sistema, e adquirir estimativas técnicas para o esforço exigido para implementar cada estória. Estar priorizando estórias pode ser o mesmo que priorizar os casos de uso da UML utilizados no RUP.

Algumas das estórias de exemplo de literatura de XP não são realmente casos de uso, assim pode não fazer sentido comparar os dois. Uma estória descreve uma unidade de trabalho, e XP assume que o contexto da estória é óbvio.

Um caso de uso provê um jogo completo de operações que provêem valor a um usuário de sistema. Pode se dizer então que as estórias e casos de uso complementam um ao outro, e que um caso de uso pode ser realizado por múltiplas estórias. Um caso de uso envolve todas as pessoas do projeto, enquanto estórias envolvem em mais detalhes com desenvolvedores.

As estórias podem ser consideradas promessas para as conversas entre o cliente e o programador. Estas conversações são de grande valor, e o RUP especificamente lhe pede para capturar esses resultados em casos de uso e outros artefatos que são

exigidos. O XP insinua que se deveria capturar os resultados mas deveria prover uma orientação em como fazer isto. Em XP, o lugar final para documentar requisitos ou decisões de projeto é o código.

Infelizmente, código não é um meio de comunicação efetivo para todos os stakeholders do projeto.

Estimativas técnicas adquiridas através do desenvolvedor que implementará uma característica no futuro é uma boa prática. O RUP não entra em detalhe de como obter estas estimativas, mas caso exista confiança no desenvolvedor, então pode adotar esta prática como parte do processo de planejamento. Na realidade, ao se entrar nos detalhes de resultados do projeto, estimativas para documentação, treinamento, apoio e desenvolvimento são feitas[18].

5.3.2 - Pequenos lançamentos

Dependendo de como é analisado o RUP e XP podem parecer bastante semelhantes nos conceitos de um lançamento[18]. O RUP define um lançamento como "uma versão estável, versão executável de produto, junto com qualquer artefato necessário para usar este lançamento, como notas de liberação ou instalação, instruções". Além disso, de acordo com o RUP, lançamentos são ou internos (não liberados para o usuário final) ou externos (liberado para o usuário final) [2].

XP define um lançamento como "uma pilha de estórias que junto fazem sentido empresarial"[10]. A prática de pequenos lançamentos parece coincidir com a prática de integração contínua. Se for interpretado que as estórias podem significar não só o código como também qualquer artefato necessário à liberação, e aceitar-se o lançamento como interno ou externo, então RUP e XP, possuem o conceito de lançamentos quase idênticos.

O RUP considera mais que só código. Um lançamento, especialmente um externo para o cliente, pode ser inútil a menos que esteja acompanhado por notas de lançamento, documentação, e treinamento.

O XP prioriza o código e assume o resto se aparecer, já que o código é o artefato primário de XP, os outros precisam ser derivados disto. Isto insinua certas habilidades que não podem ser óbvias. Por exemplo, programadores poderiam precisar ler o código para entender como funciona o sistema para produzir a documentação.[7]

Pode se assumir também que os lançamentos externos de XP são tudo que deve ser entregue a um cliente externo. Na realidade, XP não é claro sobre isto.[18]

Quando se é incapaz de entregar um sistema ao cliente, poderia-se considerar outros modos de avaliação, como prova de avaliação. Em um projeto RUP, tipicamente entrega-se o sistema como bem ao cliente na última iteração de construção e nas iterações de fase de transição enquanto no XP cada iteração termina com uma entrega de um executável ao cliente[10].

5.3.3 - Metáfora de sistema

XP usa uma metáfora para o sistema e o RUP usa uma arquitetura formal. Esta metáfora de sistema é uma estória compartilhada simples de como o sistema trabalha. Esta estória envolve tipicamente várias classes e padrões dos que moldam o sistema que está sendo construído. Baseado em comparações com algo familiar, padrões nos ajudam a entender algo novo e pouco conhecido.

A metáfora de sistema de XP pode ser uma substituição satisfatória para a arquitetura em alguns casos, mas normalmente só em sistemas pequenos. Para muitos, se não a maioria dos sistemas de software, precisa-se mais que uma simples estória.[19]

O RUP já oferece um contraste, é um processo centrado na arquitetura [6]. A arquitetura é mais que uma metáfora, embora possa incluir várias metáforas. A arquitetura está preocupada com a estrutura, comportamento, contexto, uso, funcionalidade, desempenho, reutilização, abrangência, restrições, possíveis alterações, e estéticas. Normalmente não é possível capturar tudo isto dentro uma

simples metáfora. Arquitetura não provê uma representação completa do sistema inteiro. Se concentra no que é arquiteturalmente significante e importante reduzindo riscos.

O RUP provê com riqueza orientações de como construir e administrar a arquitetura. Ajuda o analista a construir visões diferentes da arquitetura para propósitos diferentes.[6] Visões diferentes são necessárias porque há aspectos diferentes que precisam ser realçados e pessoas diferentes que precisam ver a arquitetura.

Um projeto RUP procurará definir logo no inicio a arquitetura. Freqüentemente um executável da arquitetura é produzido durante a Fase de Elaboração. Isto provê uma oportunidade para avaliar soluções e riscos técnicos críticos, e a arquitetura é construída durante as iterações de construção subseqüentes.

Uma arquitetura executável é uma implementação parcial do sistema, construída para mostrar funções e propriedades selecionadas do sistema, que em geral satisfazem requisitos não funcionais. É construída durante a fase de elaboração para reduzir riscos relacionados a desempenho, processamento, capacidade, confiança etc, de forma que o completo funcionamento pode ser somado à capacidade do sistema na fase de construção em uma fundação sólida.[9]

5.3.4 - Projeto Simples

O XP descreve que se construa o sistema simples que satisfaça os requisitos atuais, não vislumbrando características que ainda não foram solicitadas. Isto significa que devem ser implementadas quando forem realmente necessárias e não quando se percebe que isso poderia ser necessário no futuro.

O RUP descreve da mesma maneira mas com palavras diferentes: Administre suas exigências, continuamente priorize, e avalie progresso. Bem definidos, priorizados os requisitos simplificam a decisão do desenvolvedor sobre o que fazer.

O RUP também encoraja o uso de componentes e os modelos da UML para ajudar administrar a complexidade de projeto.

É equivocada a visão que em XP não se tenha que prestar atenção à infra-estrutura e arquitetura. Mas projeto simples não significa que se pode ignorar a infra-estrutura exigida ou a arquitetura. Há uma diferença grande entre RUP e XP nesta área. Já que um dos pilares do RUP é a arquitetura[18].

5.3.5 - Teste

Criar os testes antes do código é uma das regras de XP, outra é que o cliente provê o teste de aceitação. Programadores escrevem testes unitários e asseguram que o código faz o que é exigido. Clientes escrevem teste de aceitação para assegurar que o sistema faz o que é suposto que ele faça. O RUP tem uma estrutura para testar e provê orientação em como escrever testes efetivos.

Além de teste unitário e teste de aceitação podem ser requeridos: por exemplo, testes de carga etc. Combinando o RUP e XP, pode ser adquirido um foco de qualidade excelente para o software.

Em XP, a equipe de desenvolvimento usa os resultados de teste para decidir se o sistema está pronto para ser entregue ao cliente. Se o sistema passa por todos os testes de aceitação, então, o software está pronto. O RUP sugere outros critérios de aceitação além de testar. Dependendo do projeto, poderia-se considerar o treinamento do cliente, instalação, documentação, e vários outros artigos em seus critérios de aceitação de produto. Simplesmente porque o sistema passou em alguns testes não asseguram que um programador (ou o par de programadores) não inseriu uma armadilha em seu software. Às vezes, dependendo do tipo de sistema, onde é necessário código mais rigoroso, inspeções por auditores independentes são necessárias[18]

5.3.6 - Reconstrução

A reconstrução pode ser considerada também uma técnica para manter o projeto simples. O RUP não diz para ser feito dessa forma, mas não é contra, porém alerta que a reconstrução pode criar um risco para a equipe. O que é simples para um programador pode ser complexo para outro. Se muita reconstrução é feita, então pode ser que esteja gastando um tempo valioso apenas reescrevendo o que já estava escrito. O XP por sua vez incentiva à reconstrução como forma de manter o código simples ao longo das manutenções e do ciclo de vida do projeto [18]

5.3.7 - Programação aos pares

Há evidência que programação aos pares é um modo efetivo de melhorar a produtividade do programador. Os programadores permanecem mais concentrados, e por eles adquirem avaliação imediata à qualidade melhora. Programação aos pares é um modo para evitar revisões de código, mas coloca algumas restrições a equipe do projeto:

- a equipe deve estar no mesmo local de trabalho; e
- os pares devem ter personalidades e habilidades compatíveis.

O XP tem como base no seu processo a programação aos pares para aumentar a produtividade e a qualidade do produto gerado. O RUP por sua vez não o explicita e como é um processo que abrange vários tipos e tamanhos de projetos não é sempre que isto pode ser aplicado[19].

5.3.8 - Propriedade coletiva

Em XP todos os membros da equipe de desenvolvimento têm permissão para fazer mudanças a qualquer parte do código e tem a responsabilidade de fazê-las.

Há um benefício óbvio com esta prática. Quando se necessita mudar o código, pode se mudar e pode com isto ganhar tempo em seu trabalho sem ter que esperar que outra pessoa o faça. Porém para que esta prática funcione é necessário também fazer integração contínua e manter uma atividade intensa de testes. Se qualquer código

mudar, então é necessário fazer os testes e não se considerar terminada a mudança até toda a passagem de testes.

Mas esta propriedade coletiva pode não funcionar bem em todos os projetos. Grandes sistemas contêm muito conteúdo para uma única pessoa entender tudo a um nível detalhado. Alguns sistemas pequenos incluem freqüentemente código que é complexo devido a seu domínio ou a função executada. Se um especialista é requerido, então propriedade coletiva pode não ser apropriada.

Quando um sistema é desenvolvido em um ambiente distribuído, não é possível que todo o mundo para modifique o código. Nestes casos, XP oferece uma prática apoiando chamada "*code stewardship*". [12] Nesta prática, uma pessoa, tem a responsabilidade do código, com contribuição de outros membros da equipe. Todos podem alterar o este trecho do código porém a pessoa escolhida tem responsabilidade de verificá-lo. Não existe nenhuma diretriz de quando aplicar esta prática em vez de propriedade coletiva.

Propriedade de código coletiva provê um modo para a equipe mudar código rapidamente quando precisa mudar. A desvantagem potencial nisto é que todo o código é mudado, então há alguns ítens que podem ser necessários controlar de um modo centralizado, por exemplo, quando o código é modificado porque falta um pouco de funcionalidade. Se um programador está implementando uma estória (ou um caso de uso, ou um cenário), e requer comportamento de uma classe, a propriedade de código coletiva permite a classe ser modificada naquele mesmo lugar. Contanto que o sistema seja pequeno bastante para um programador entender tudo do código, isto poderia funcionar bem. Mas como o sistema se torna maior, é possível que a mesma funcionalidade possa ser somada para codificar e isso exista em outro lugar. Esta redundância poderia ser pega a algum ponto e o código reconstruído, mas é certamente possível que o código seja alterado e a funcionalidade comece a apresentar problema em outra funcionalidade do sistema.

A propriedade coletiva de código pode começar a ser utilizada em um projeto, contanto que se tenha boa administração de código e ferramentas efetivas, que então trabalharão até certo ponto. O líder de projeto ou gerente precisa controlar quando a base de código fica muito grande ou muito especializada.

Quando isto acontecer, pode ser necessário estruturar o sistema em um conjunto apropriado de componentes e subsistema e assegurar quais membros específicos da equipe são responsáveis por eles. O RUP provê orientação de como estruturar o sistema.

Pelos problemas citados acima O RUP não é favorável à propriedade de código coletiva. Ao contrário propriedade coletiva é um dos princípios de XP.[9]

5.3.9 - Integração contínua

Todo programador em um projeto XP deve poder alterar o código e assegurar o trabalho, não só por testes unitários, mas também por testes de aceitação. Isto requer freqüentes integrações: uma ou mais por dia. Para que isto realmente possa ocorrer é necessário uma administração de configuração poderosa, de ferramentas e um processo efetivo para ser usado.

Este ponto está de acordo no RUP e XP. Porém o RUP provê diretrizes para integração contínua como também informação específica para o uso de ferramentas.[19]

5.3.10 – Quarenta (40) horas de trabalho semanal

Estudos indicam que a maioria das pessoas diminuem o rendimento rapidamente e consideravelmente quando eles investem mais de quarenta horas no trabalho, especialmente quando é habitual. Um projeto XP proíbe duas semanas sucessivas de trabalho extra. O RUP não proíbe explicitamente como o XP, porém concorda com esta prática.

5.3.11 - Cliente dedicado

Originalmente, o XP descreve que um cliente tem que sentar com a equipe, disponível para perguntas e respostas, solucionando disputas, e planejando as prioridades em pequena escala. Isto foi refinado mais adiante para, um projeto de XP é guiado por um individuo dedicado que é autorizado a determinar requisitos, prioridades, respostas às perguntas dos programadores.

O RUP é mais flexível. Embora sempre manteve que o cliente, na realidade todos os *stakeholders*, devem ser representados adequadamente no projeto, o RUP também reconhece que não é sempre possível ou desejável ter um cliente real situado com a equipe de desenvolvimento. Ao invés disto, RUP define vários papéis que são responsáveis por determinar as metas de projeto, extensão, e assim por diante, e diz que um cliente (no local ou não) ou alguma outra pessoa apropriada na organização pode executar as atividades traçadas a estes papéis. Não é importante se a pessoa é um cliente atual, ou se ele ou ela está de fato no local. O que é importante é que a pessoa esteja disponível para esclarecer dúvidas e seja responsável o suficiente para produzir a informação necessária para a equipe progredir tão depressa quanto possível.

5.3.12 - Código Padrão

O RUP e XP concordam que é necessário se ter códigos padrões, mas que mais importante que ter estes padrões é utilizá-los.

5.3.13 – Resumo

Com a Tabela 1 é possível verificar o resumo das comparações dos 12 princípios, indicando o quanto RUP concorda com os princípios praticados por XP.

Tabela 1 – Resultado da comparação RUP com XP.

Princípios do XP	XP	RUP
Planejamento	Apóia esta pratica.	Apóia esta pratica.
Pequenos lançamentos	Define como resultado final para o lançamento o código.	Considera mais que o código no lançamento.
Metáforas do Sistema	Utilizado para definir a arquitetura. Não considera uma arquitetura formal	É centrado na arquitetura.
Projeto Simples	Apóia esta pratica.	Apóia esta pratica.
Teste	Apóia esta pratica.	Apóia esta pratica
Reconstrução	Apóia esta pratica	Alerta para o risco adicionado ao projeto.
Programação aos pares	Apóia esta pratica.	Alerta que são em todos que pode ser aplicado
Propriedade coletiva	Apóia esta pratica.	Considera que nem todos os membros da equipe estão aptos a alterar qualquer parte do código.
Integração contínua	Apóia esta pratica.	Apóia esta pratica
Quarenta horas de trabalho semanal	Apóia esta pratica.	Concorda mas não proíbe explicitamente.
Cliente dedicado	Apóia esta pratica.	O cliente não precisa estar dedicado, mas alguém que o represente.
Código padrão	Apóia esta pratica.	Apóia esta pratica.

6 - CONCLUSÕES

XP é focado em desenvolver o código, ajuda a reduzir os riscos e assegura que é possível entregar o produto certo ao cliente no prazo certo. Porém, quando se olha para um projeto de desenvolvimento com um conjunto completo de entregáveis, código, documentação, treinamento, e apoio, há muitas coisas no RUP que não são considerados em XP. Com isto é preciso determinar o que é necessário para o projeto onde a metodologia será aplicada. Segue abaixo alguns dos itens que devem ser considerados.

- Modelagem de Negócio. O assunto inteiro de modelagem de negócio é ausente em XP. São construídos sistemas em uma organização, o conhecimento da organização pode ser importante ao identificar os requisitos e para entender como a solução pode ser aceita;
- Inicio do projeto. XP assume que o projeto já está justificado e não diz como a justificativa acontece. Em muitas organizações, um caso empresarial deve ser feito antes de um projeto sério começar. O RUP ajuda a equipe a fazer seu caso empresarial por desenvolver visões para todas as pessoas importantes para o projeto; e
- Entrega. XP prioriza o código e a entrega para ele é apenas o sistema funcionando e não prioriza materiais e apóio, documentação de apoio etc. Porém a maioria dos projetos necessitam destes artefatos. Produtos de software comerciais, por exemplo, requerem empacotamento, distribuição, manuais de usuário, materiais de treinando, e uma organização de apoio. O RUP provê uma orientação de como criar materiais apropriados.

A diversidade de processo é importante. Um único formato não se ajusta a todos os projetos. O processo que é usado para o projeto deve ser apropriado para ele. Considera-se que é necessário adotar a melhor forma para o projeto. Consideram-se todos os aspectos e riscos. Usa-se tudo que é necessário, mas balancea-se para se ter o peso ideal para o projeto em questão.

O RUP e XP provêem duas aproximações diferentes de desenvolvimento de projetos de software. Eles complementam um ao outro de vários modos.

O XP se concentra em código e técnicas para uma equipe pequena criar código. Dá ênfase à comunicação de interpessoal e investe o mínimo esforço em artefatos que não são o código.

O RUP é uma estrutura de processo que pode ser configurado para tipos diferentes de projetos. Consideram-se riscos e técnicas de diminuição de riscos. O RUP é interpretado mal freqüentemente como sendo pesado e burocratizado porque, como uma estrutura, ele provê informação de processos para muitos tipos de projetos. Na realidade, um exemplo configurado de RUP pode ser muito leve, dependendo dos risco e do projeto. Pode incorporar algumas das excelentes técnicas de XP e outros processos se eles são apropriados para o projeto.

6.1 - Contribuições do trabalho

A escolha e a aplicação do processo adequado de desenvolvimento afeta diretamente a qualidade e o sucesso dos projetos. É necessário avaliar as características do projeto e da organização para que isto seja feito da maneira mais adequada.

Conhecer e avaliar os processos existentes identificando suas características, diferenças, melhores práticas e como podem ser otimizados ajudam os gerentes e interessados a decidir qual o melhor processo a ser utilizado.

O RUP é um processo mais completo e adaptável a vários tipos de projetos e organizações, enquanto o XP é mais adequado a projetos com características específicas. Porém pode-se notar que eles não são excludentes, isto é, é possível se escolher um dos dois processos e aplicar práticas do outro para se chegar a um processo mais otimizado e adequado para cada organização.

6.2 - Trabalhos futuros

Outros trabalhos podem ser iniciados a partir desta monografia como descritos abaixo:

- Aprofundar a comparação de RUP e XP em relação aos 12 princípios

- Outras comparações analisando fases, artefatos e envolvidos
- Estudar como XP pode ser adequado a projetos maiores e de maior complexidade utilizando-se de artefatos e técnicas do RUP.

7 – REFERÊNCIAS BIBLIOGRÁFICAS

[1] THE STANDISH GROUP REPORT."CHAOS", 1995.
Disponível em: <http://www.scs.carleton.ca/~beau/PM/Standish-Report.html>.
Acesso em: outubro/2003.

[2] PHILIPE KRUCHTEN. Rational Unified Process—An Introduction. 2 Edição. Addison Wesley,2000. 320p.

[3] BARRY W. BOEHM. USC Anchoring the Software Process, novembro 1995. Disponível em: <http://sunset.usc.edu/publications/TECHRPTS/1995/usccse95-507/ASP.pdf> Acesso em: outubro/2003.

[4] IVAR JACOBSON; GRADY BOOCH;JIM RUMBAUGH. Unified Software Development Process. 1 Ediçao. Addison Wesley, 1999. 512p.

[5] GRADY BOOCH; JIM RUMBAUGH; IVAR JACOBSON. Unified Modeling Language—User's Guide. 1 Edição. Addison Wesley, 1999. 490p.

[6] PHILIPE KRUCTHEN. Architetural Blueprints the 4+1 view model of architeture, novembro 1995. Disponível em: <www.rational.com/media/whitepapers/Pbk4p1.pdf>. Acesso em: novembro de 2003.

[7] BECK, KENT. Extreme Programming Explained: Embrace Change. 1 Edicao. Addison Wesley, 1999. 224p.

[8] ALAN RADDING. Radical simplicity, but whith control, abril 2001 Disponível em: http://www.informationweek.com/831/appdev.htm. Acesso em: outubro 2003

[9] GARY POLLICE, RUP and XP Part II: Valuing Differences, 2001. Disponível em:<http://www.therationaledge.com/content/apr_01/f_xp2_gp.html> Acessado em: novembro de 2003

[10] CORSARO, ANGELO. Extreme Programming Concepts, abril 2001. Disponível em:<http://tao.doc.wustl.edu/~corsaro/resources/papers/XPConcepts.pdf> Acessado em: outubro de 2003.

[11] MARTIN FOWLER. Contínuos integration. Disponível em <http://www.martinfowler.com/articles/continuousIntegration.html> Acessado em novembro 2003

[12] DAVE SMITH.. Code StewardShip. Disponível em: <http://c2.com/cgi/wiki?CodeStewardship>. Acessado em: outubro de 2003.

[13] BACH, J. Enough about process: What we need are heroes. IEEE Software, v. 12, n. 2, p. 96-98, Março 1994.

[14] JIM HEUMAN, Introduction to business modeling using the UML. Disponível em: <http://www.therationaledge.com/content/mar_01/m_uml_jh.html> Acessado em: novembro de 2003

[15] JOHN SMITH, A Comparison of Rup and XP,2001. Disponível em: <http://www.rational.com/products/whitepapers/423.jsp> Acessado em: novembro de 2003

[16] OGNIAN PISHE. Notes on a Practical Guide and Thoughts on Software Development. Journal of Object Technology, ETH Zurich, Vol 1, N°2, julho-agosto, 2002.

[17] ROBERT L. GLASS, extreme Programming: The Good, the Bad, and the Bottom Line,2001. Disponível em: <www2.umassd.edu/SWPI/xp/papers/IEEESWNov2001/s6112.pdf> Acessado em: novembro de 2003

[18] GARY POLLICE, RUP and XP, Part I: Finding Common Ground, 2001. Disponível em:<http://www.therationaledge.com/content/mar_01/f_xp_gp.html> Acessado em: novembro de 2003

[19] RATIONAL SOFTWARE CORPORATION. Concept: Agile Practices and RUP. Disponível em: <http://latitude.east.asu.edu/494/rup/RationalUnifiedProcess/process/workflow/environm/co_agile.htm>. Acessado em novembro de 2003.